JavaScript, C#, Angular, Machine Learning, AI, Security

CODE

NOV
DEC
2025

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 8.95 Can $ 11.95

Integrating AI and ML in .NET Core with ML.NET

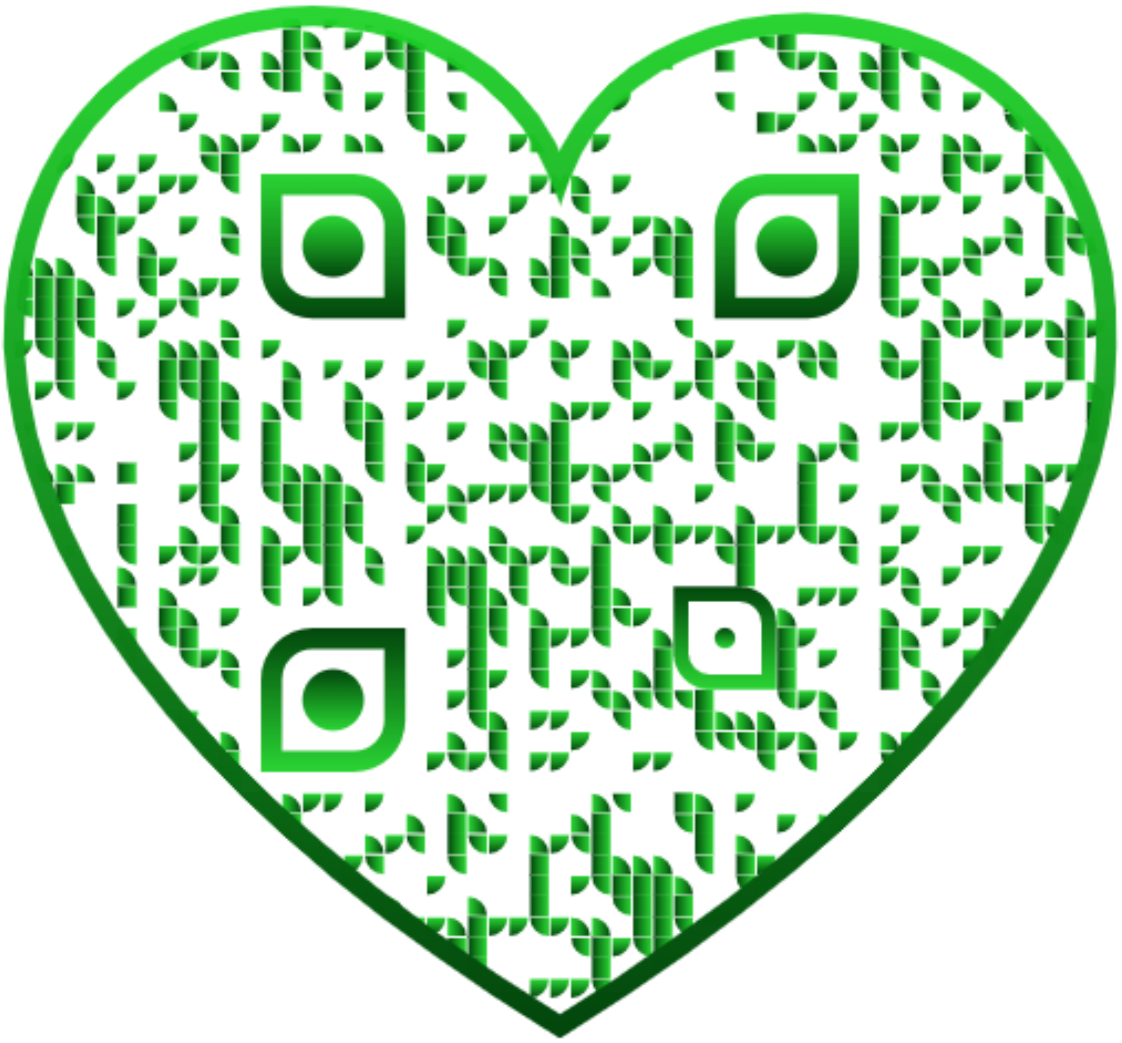
Cover AI generated - Markus Egger

Add Custom Scripting to Your Apps

Understand Model Context Protocol (MCP)

JavaScript Classes 101

1 1>
7 25274 56130 8



CODE
CONSULTING

ARE YOU WONDERING HOW ARTIFICIAL INTELLIGENCE CAN BENEFIT YOU TODAY?

©shutterstock

EXECUTIVE BRIEFINGS

Are you wondering how AI can help your business? Do you worry about privacy or regulatory issues stopping you from using AI to its fullest? We have the answers! Our Executive Briefings provide guidance and concrete advise that help decision makers move forward in this rapidly changing Age of Artificial Intelligence and Copilots!

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

AI-SEARCHABLE KNOWLEDGEBASE AND DOCUMENTS

A great first step into the world of Generative Artificial Intelligence, Large Language Models (LLMs), and GPT is to create an AI that provides your staff or clients access to your institutional knowledge, documentation, and data through an AI-searchable knowledgebase. We can help you implement a first system in a matter of days in a fashion that is secure and individualized to each user. Your data remains yours! Answers provided by the AI are grounded in your own information and is thus correct and applicable.

COPILOTS FOR YOUR OWN APPS

Applications without Copilots are now legacy!

But fear not! We can help you build Copilot features into your applications in a secure and integrated fashion.

CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.

codemag.com/ai-services

832-717-4445 ext. 9 • info@codemag.com

Features

8 Add Some Class to Your JavaScript

Paul introduces the ES6 JavaScript class keyword and shows you how to pass in arguments, create public read-only properties, and make private fields. You'll learn how to override an inherited method in an extended class, call methods in the parent class, and extend the built-in classes.

Paul Sheriff

20 Threat Detection Using Anomaly Detection

You might be most successful fighting off AI security attacks by using AI to identify anomalies, according to Sahil. Follow along as he builds a local anomaly detection system using Python.

Sahil Malik

26 Angular Signals in the Real World: Smarter Inputs and Reactive Routing

In this second installation of his series on Angular Signals, Sonu designs Angular Signals applications that replace lifecycle-bound input handling, remove boilerplate queries, update two-way binding, use signals and route parameters to improve navigation decisions, and connect to signal-powered API calls.

Sonu Kapoor

32 Integrating AI and ML in .NET Core with ML.NET

Joydip looks at how Microsoft's cross-platform, open-source machine learning framework, ML.NET, brings AI and ML to .NET developers. You can build, train, and deploy models that get your business where you want it to go using tools already available in .NET.

Joydip Kanjilal

56 Custom Scripting for Web Applications

Nevio, Enzo, and Vassili explore using a customized C# scripting language to develop server-side API scripts for web applications.

Nevio Medancic, Enzo Grubisa, and Vassili Kaplan

65 MCP: Building the Bridge Between AI and the Real World

Despite AI's current unwieldiness, Jason is determined to wrangle it into submission. Using Model Context Protocol (MCP), he shows you how to establish a stable infrastructure and build the tools you need, with request context, authorized sources, and security.

Jason Murphy

70 Fixing Security Issues in Seconds, Not Sprints, Powered by GenAI

Security is always on everyone's minds, but it often comes too late in the development process to be handled quickly or comprehensively. Gaurav uses SecureCodeAgent, a GenAI-powered solution, earlier in the development cycle.

Gaurav Mittal

Departments

6 Editorial

18 Advertisers Index

74 Code Compilers

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay $50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact Customer Service at 832-717-4445 ext. 9.

Subscribe online at www.codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A.

BrickCon

BRICKCON is Coming to Orlando, Florida!

Join us at **BRICKCON**, a Databricks Community Conference tailored for data professionals of all kinds – from data engineers to MLOps specialists, data analysts, and AI/ML practitioners.

Mark your calendars for **Dec. 1-5, 2025**
Location: **Walt Disney Dolphin Resort in Orlando, Florida**

We already have a stellar lineup of Databricks thought leaders and MVPs, including:

Simon Whitely

Denny Lee

Dustin Vannoy

Adi Polak

Scott Haines

Douglas Leal

...and more!

Stay **Tuned.**

Updates on parties, networking events, and exciting opportunities to connect, learn, and grow together at BRICKCON. See you there!

brickcon.ai

Reading for Fun and Profit

One of my favorite memories of community college was sitting in my professors' offices admiring their extensive book collections. These tomes of technology were stacked to the ceiling and I was jealous. "Someday, I'll possess a library to rival these profs. Someday!" I told myself. Well that someday came to pass and eventually I ended up with an extensive collection of books written by legends of the technological world. Yourdon, DeMarco, Brooks, The Gang of Four. I had them all, and over the years, these books have enhanced my coding in numerous ways. Recently I purged my collection and focused on my favorite books with an eye on the ones that are timeless to me. By "timeless" I mean that these are books that I return to again and again because they provide inspiration.

I thought it would be fun to highlight some of the top books in my collection and why I kept them. In no particular order, here they are.

"The Psychology of Computer Programming," Gerald M. Weinberg, Van Nostrand Reinhold, 1971. I recall finding this book in Seattle sometime in the late 90s in a used bookstore. This find had a great impact on my career. This book had many chapters and sections dedicated to building software teams and how building software is social in nature. I see a lot of Agile concepts that may have sprung from these pages. For instance, there's a section on formal and informal organization.

"The Mythical Man Month," Frederick P. Brooks, Jr., Addison-Wesley, 1974. This is probably the most well-known book in my collection and tells many stories of the pitfalls of software development and lessons on how to avoid them. He talks about team organization, the effects of adding more developers to a project, and how there can be an inverse correlation between the number of developers and building code.

"Postmortems from Game Developer," Austin Grossman, CMP Books, 2003. After 30+ years in the software industry, the closest I ever came to creating a game was writing a program to calculate the amount in the case on the Deal or No Deal show. This being the case, the pages of Game Developer Magazine informed a lot my software practices for years. The "Postmortem" columns were my absolute favorites. The postmortems were fun reads because they discussed things that went right and wrong in real-world software projects, e.g., games. Even though they were games, the problems were similar to business development: ambitious schedules, new technology, unclear requirements, and, like Cheech and Chong said, "Things are tough all over."

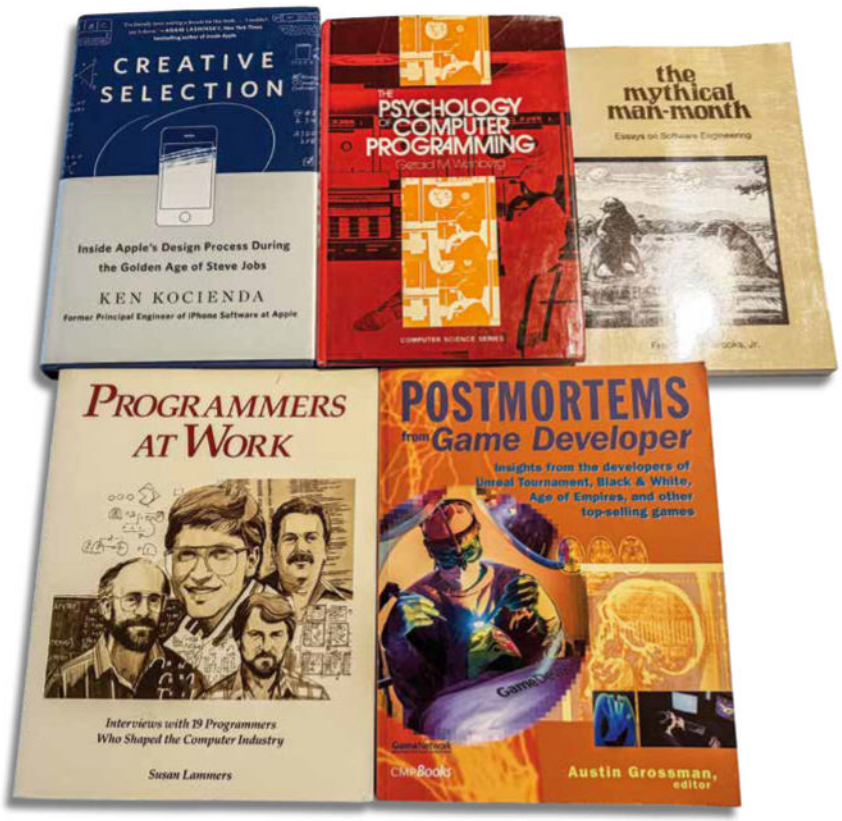

CREATIVE SELECTION
Inside Apple's Design Process During the Golden Age of Steve Jobs
KEN KOCIENDA
Former Principal Engineer of iPhone Software at Apple
THE PSYCHOLOGY OF COMPUTER PROGRAMMING
the mythical man-month
Essays on Software Engineering
PROGRAMMERS AT WORK
Interviews with 19 Programmers Who Shaped the Computer Industry
Susan Lammers
POSTMORTEMS from Game Developer
Insights from the developers of Unreal Tournament, Black & White, Age of Empires, and other top-selling games
CMPBooks
Austin Grossman, editor

Figure 1: Some good books

"Creative Selection," Ken Kocienda, St Martin's Press, 2018. In a vein very similar to the postmortems book, "Creative Selection" tells stories of building two of the most used pieces of software on the planet: the Safari browser and the iPhone keyboard. Yes, you read that right. The iPhone keyboard is one of the most used pieces of software in the world and this person was on a team that built it. The stories in this book document what it's really like to build software from concept to production. The hardships, false starts, successes, and setbacks are all there, in stories that we can all imagine because we all use this class of software in one form or another

"Programmers at Work," Susan Lammers, Microsoft Press, 1986, 1989. This book is a pure joy because it has interviews with the programmers responsible for some of our favorite programs. This book has interviews with people like Bill Gates and Gary Killdal of MS-DOS and CP/M operating systems. And John Warnock, Dan Bricklin, and Bob Frankston, who created Adobe and VisiCalc, among other tools. A lot of these interviews have actual code samples, design, and diagrams from the creation of some of our favorite apps. Oh, and for the FoxPro folks that still read these pages, there's an interview with C. Wayns Ratliff, creator of dBase II.

When I was putting this together, I spent some time reading a few sections here and there, and now I remember why I kept the tomes I did. These books are timeless! Tell me about your favorite resource books.

Rod Paddock
CODE

CODE CONSULTING

ARE YOU WONDERING HOW ARTIFICIAL INTELLIGENCE CAN HELP YOUR BUSINESS?

Do you worry about privacy or regulatory issues stopping you from using AI to its fullest?

We have the answers!

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.

codemag.com/executivebriefing

832-717-4445 ext. 9 • info@codemag.com

©shutterstock

Add Some Class to Your JavaScript

Many of us have been using JavaScript for many, many years. Although you may know most of the language inside and out, there have been many advancements since June of 2015 with the release of ECMAScript Version 6 (ES6). One of the most exciting features added to the JavaScript language is the ability to use the class keyword to create classes. If you're familiar with C# or other object-oriented languages, classes are a great way to keep your code encapsulated, organized, maintainable, testable, and reusable. You now have those same capabilities in JavaScript.

Paul D. Sheriff
www.pdsa.com

Paul has been working in the IT industry since 1985. In that time, he has successfully assisted hundreds of companies' architect software applications to solve their toughest business problems. Paul has been a teacher and mentor through various mediums such as video courses, blogs, articles and speaking engagements at user groups and conferences around the world. Paul has multiple courses in the www.pluralsight.com library (https://bit.ly/3gvXgvj) and on YouTube.com (https://www.youtube.com/@pauldsheriff) on topics ranging from C#, LINQ, JavaScript, Angular, MVC, WPF, XML, jQuery, and Bootstrap. Contact Paul at psheriff@pdsa.com.

In this article, you're introduced to the ES6 JavaScript **class** keyword. You learn to create classes with constructors into which you may or may not pass in arguments. You learn to create public, read-only, and write-only properties. Within a class, you can create private fields and methods that are only available within the class itself. Inheritance is a key feature of object-oriented programming and is accomplished in JavaScript using the **extends** keyword. You learn to override a method in an extended class and how to call methods in the parent class. You even learn how to extend the built-in classes in JavaScript. Finally, you build utility classes using static properties and methods.

Create Your First JavaScript Class

Classes are created within either .html or .js files. Obviously, placing classes into a .js file makes these classes reusable. To get started, create a folder named **MovieSamples** somewhere on your hard drive. Within this folder, add a folder named **classes**. Within the **classes** folder, add a new file named **Movie.js** into which you add the JavaScript shown in the following code snippet. Feel free to use VS Code, Visual Studio, or any other editor in which you are comfortable to follow along with this article.

```
class Movie {
  // Public Properties
  filmId;
  name;
  genres;
  actors;
}
```

The code starts with the **class** keyword followed by the name of the class. Just like in most object-oriented languages, you should use an upper-case letter for the class name. Within the curly braces, you define public properties by using the name of the property terminated with a semicolon. The Movie class defined in this code snippet has four public properties: **filmId, name, genres,** and **actors**. These properties hold the data used to describe a movie.

Use the Movie Class

In the **MovieSamples** folder, create a file named **class.html** and add to this new file the code shown in **Listing 1**. To create a new instance of the Movie class so you can place data into those properties, write the following line of code just below the **use strict** line within the <script> element.

```
let movie = new Movie();
```

The **new** keyword creates an instance of the Movie class and allocates memory into which you can assign data to the properties. The **movie** variable holds the reference to the Movie object in memory. You assign data through the **movie** variable to each property using dot notation, as shown in the following code snippet.

```
movie.filmId = 1;
movie.name = "Young Frankenstein";
movie.genres = "Comedy";
movie.actors = "Gene Wilder,Marty Feldman";
```

Assign values to each property by using the variable name, **movie**, followed immediately by a period (.), followed by the name of one of the public properties. On the right-hand side of the equal sign is where you specify the value to assign to each property. The movie object in memory is now holding onto this data until you wish to use it. For example, you can write the following lines of code to display each property on the console window.

```
console.log(`Film ID: ${movie.filmId}`);
console.log(`Name: ${movie.name}`);
console.log(`Genres: ${movie.genres}`);
console.log(`Actors: ${movie.actors}`);
```

Try It Out

Display the **class.html** file in your browser, open your F12 Developer Tools and click on the Console tab (**Figure 1**) to view the results of the code that has run.

Alternative Ways to Declare a Class

The method I just showed to declare a new class is the most common and the preferred method. However, JavaScript does allow a couple other methods of declaring a class. An alternative way to define a class is to use the

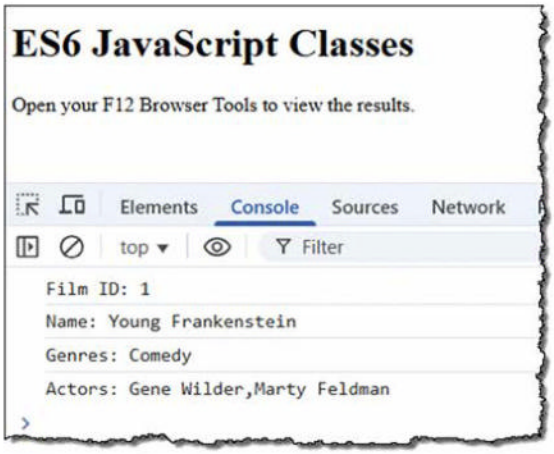


Figure 1: Display each property value in the console window.

class keyword with no name and assign the class name using the **let** statement. This declaration looks like the following code snippet:

```
// Can leave off the name of the class
let Movie = class {
  // Public Properties
  filmId;
  name;
  genres;
  actors;
}
```

This syntax looks like how you create an object literal, but because of the **class** keyword you cannot use the **Movie** variable without first creating an instance. You must still use the **new** keyword just like you did in the first example.

```
let movie = new Movie();
```

Define a Class with a Different Name

Another way to define a class is to use the **class** keyword with a name after it, but you then assign a different name using the **let** statement, as shown in the following code:

```
let Movie = class MovieClass {
  filmId;
  name;
  genres;
  actors;
}
```

This form of declaration is assigning an alias to the MovieClass so it's always known as **Movie**. The **MovieClass** name is not useable anywhere in your program. For example, the following code does NOT work:

```
// NOTE: This does NOT work!
let movie = new MovieClass();
```

I can't really think of a use case for this syntax, but it's valid code in JavaScript.

Classes Are Not Hoisted

In JavaScript, code is executed top to bottom. This typically means that you must create a variable or function declaration prior to calling that function or using that variable. However, you're allowed to call functions or use variables, even when the functions/variables are located further down in your <script> tag, as illustrated in the following code:

```
<script>
  displayMovie("Young Frankenstein");

  function displayMovie(name) {
    console.log(name);
  }
</script>
```

The above code works just fine because JavaScript performs what is called "hoisting." This means JavaScript automatically moves function and variable declarations to the top of their scope. Functions and variables are moved by the interpreter to be above where they are called from. JavaScript, however, does NOT hoist class definitions. This means that you must always declare classes prior to using them. Not really a big deal, but it's something you must be aware of.

Listing 1: Create an HTML page to try out the functionality of JavaScript classes.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">

  <title>ES6 JavaScript Classes</title>
</head>

<body>
  <header>
    <h1>ES6 JavaScript Classes</h1>
  </header>

  <main>
    <p>
      Open your F12 Browser Tools to view the results.
    </p>
  </main>

  <script src="classes/Movie.js"></script>
  <script>
    'use strict';


  </script>
</body>

</html>
```

Constructors in Classes

A constructor is a method that runs when the class is instantiated using the **new** keyword. A constructor is created using the constructor() function name followed by any code between curly braces. The most common use of a constructor is to initialize any internal properties or fields to a default value. In the following code, you can see each of the public properties being assigned to either a zero or an empty string:

```
class Movie {
  // Public Properties
  filmId;
  name;
  genres;
  actors;

  constructor() {
    this.filmId = 0;
    this.name = "";
    this.genres = "";
    this.actors = "";
  }
}
```

Automatic Public Properties

In the Movie class, four public properties were explicitly created within the declaration of the class. However, you do not need to explicitly create public properties if you declare them in the constructor. Use the **this** keyword

before the property name you wish to declare, as shown in the following code:

```
class Movie {
  // Create a constructor for this class
  constructor() {
    // Automatically create public properties
    this.filmId = 0;
    this.name = "";
    this.genres = "";
    this.actors = "";
  }
}
```

This code is the equivalent of the Movie class you just added the constructor to, just without the four explicitly declared public properties. You may choose to either explicitly declare your public properties, or define them within the constructor, the choice is yours.

Try It Out

If you wish, replace the previous Movie class definition with the one presented here. Save the changes to your file and display the **class.html** file in your browser to see the same results as shown in **Figure 1**.

Pass Arguments to a Class Constructor

A constructor is nothing more than a special kind of function. Like a function, a constructor can accept one or more parameters. Open the **Movie.js** file and modify the constructor of the Movie class to accept four parameters and assign the values passed into the appropriate properties.

```
class Movie {
  constructor(id, name, genres, actors) {
    this.filmId = id;
    this.name = name;
    this.genres = genres;
    this.actors = actors;
  }
}
```

Open the **class.html** file and modify the code that creates the instance of the Movie class and pass in arguments to the constructor to place into the appropriate properties, as shown in the following code snippet. Delete the code that assigns values to each public property within this file.

```
let movie = new Movie(1,
  "Young Frankenstein", "Comedy",
  "Gene Wilder,Marty Feldman");
```

Try It Out

Display the **class.html** file in your browser and, once again, you should see the same results as shown in **Figure 1**.

Explicitly Define Class Properties

You can explicitly define the public properties in your class and still have a constructor to assign values to those properties, as shown in the following code snippet. How you choose to create your public properties either implicitly or explicitly is a matter of style. I recommend that whichever style you choose, stay consistent throughout your applications.

```
class Movie {
  // Public Property Definitions
  filmId;
  name;
  genres;
  actors;

  constructor(id, name, genres, actors) {
    this.filmId = id;
    this.name = name;
    this.genres = genres;
    this.actors = actors;
  }
}
```

Add Public Methods to a Class

Methods in a JavaScript class are simply functions without the function keyword in front of them. To access properties of the class from within methods, you must use the **this** keyword before each property name just as you did in the constructor. Open the **Movie.js** file and add the following two methods just after the constructor of the Movie class.

```
// Public Method Definitions
getMovieInfo() {
  return `Film ID: ${this.filmId}\n
          Name: ${this.name}\n
          Genres: ${this.genres}\n
          Actors: ${this.actors}`;
}

asJSON() {
  return JSON.stringify(this);
}
```

In the getMovieInfo() method, I had to split the return string onto different lines for purposes of the printed page. Be sure to remove all the extra spaces from the literal string in the getMovieInfo() method before you run this code.

Open the **class.html** file and remove the four lines of code that display each public property in the console window. Instead, make a call to the getMovieInfo() method on the **movie** object variable, as shown in the following code snippet. Add a call to the asJSON() method as well.

```
let movie = new Movie(1,
  "Young Frankenstein", "Comedy",
  "Gene Wilder,Marty Feldman");

// Display the movie data
console.log(movie.getMovieInfo());

// Display the movie data as JSON
console.log(movie.asJSON());
```

Try It Out

Save the changes to both the **Movie.js** and **class.html** files. Display the **class.html** file in your browser, open

Figure 2: Use methods to display the properties of a class in different formats.

the F12 Developer Tools, and your page should look like **Figure 2**.

Optional Chaining Operator

JavaScript returns a value of **undefined** if you try to access a property that does not exist in a class or an object literal. However, if you try to access a property on a nested object, JavaScript throws an exception. Use the optional chaining operator (?) to avoid the exception being thrown. The optional chaining operator returns a value of **undefined** on the property that is undefined.

For example, if you have an instance of the movie object and you try to access a property named releaseDate, as shown in the following code snippet, an undefined is returned.

```
console.log(movie.releaseDate);
```

Now, try to access a property under another property that does not exist, as in the following code:

```
console.log(movie.metaData.releaseDate);
```

In this case, you get an **Uncaught TypeError** thrown by JavaScript. The way to fix this is to add a question mark (?), the optional chaining operator, before the second to last property in the chain, as shown below:

```
console.log(movie.metaData?.releaseDate);
```

When this code is run, an **undefined** is returned because of the optional chaining operator. Execution stops once the name of the property or method name used just before this operator does not exist and returns an **undefined** instead of throwing an exception.

Try It Out

Add these lines of code to the **class.html** and display the page in your browser so you can see the **undefined** value and the exception. Comment out these first two lines and display the class.html file in the browser again. There should be no exception but instead, there should be an **undefined** in the console. After seeing the results, delete these three lines of code so you can continue with the rest of this article.

Class Versus a Function Constructor

JavaScript is a prototype-based language, and prior to the ES6 standard to achieve encapsulation, you created a function constructor into which you wrapped up functionality. The **class** keyword is just an abstraction over a prototype-based function constructor. For example, if you take the version of the Movie class you've written so far, you can translate this into the old way of declaring a class using a function constructor, as shown in **Listing 2**.

Listing 2: Create a class using a function constructor.

```
/* BEGIN: Movie Function
   Constructor Definition */
function Movie(id, name, genres, actors) {
  this.id = id;
  this.name = name;
  this.genres = genres;
  this.actors = actors;
}

Movie.prototype.getMovieInfo = function () {
  return `Film ID: ${this.filmId}\n
          Name: ${this.name}\n
          Genres: ${this.genres}\n
          Actors: ${this.actors}`;
}

Movie.prototype.asJSON = function () {
  return JSON.stringify(this);
}
/* END: Movie Function Constructor Definition */
```

In this code, a function constructor named Movie() is created that accepts four parameters just like you passed into the constructor of the Movie class. Use the **this** keyword to declare properties into which you place the values from the appropriate parameters. Add on methods using the **prototype** object exposed from the Movie() function object to add new methods to this class. The code between this prototype function constructor and the **class** keyword are similar, but I think the class declaration is simpler and much easier to keep self-contained.

Try It Out

Copy the **class.html** file to a new file named **function-constructor.html** and replace the **class Movie** declaration with those lines shown in **Listing 2**. Leave the rest of the lines of code in place. Save your changes and open the **function-constructor.html** file in your browser. Open the F12 Developer Tools and view the console window to see the same results you saw when you ran the class.html file.

Class Inheritance

One of the best things about object-oriented programming is the ability to inherit the properties and methods of another class and add on additional properties and methods. This technique significantly cuts down the amount of code you must write. In JavaScript, use the **extends** keyword to inherit the properties and methods from another class. Look at the following code snippet and notice the **extends** and **super** keywords.

```
// Define the class structure
class RentalMovie extends Movie {
  // Constructor Definition
  constructor(id, name, genres, actors, price) {
    // Call base class constructor
    super(id, name, genres, actors);

    // Create additional public properties
```

Listing 3: Create an HTML page to illustrate extending the JavaScript Number class.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">

  <title>
    Extend the Number Class
  </title>
</head>

<body>
  <header>
    <h1>Extend the Number Class</h1>
  </header>

  <main>
    <p>
      Open your F12 Browser Tools to
      view the results.
    </p>
  </main>

  <script>
    'use strict';

  </script>
</body>

</html>
```

```
    this.price = price;
  }
}
```

Try It Out

Add the class and the code to use the RentalMovie class to the **class.html** page and save the changes. Display the **class.html** page in your browser and you should see the movie information appear as it did before. In the console window, below the movie information, the rental price is also displayed.

Override a Class Method

If you add on a new property, you might like that property's value to appear when you call the getMovieInfo() method. You can do this by overriding the getMovieInfo() method within the **RentalMovie** class. Just after the constructor of the **RentalMovie** class, add the following code:

```
// Public Method Definitions
getMovieInfo() {
  // Call method in base class
  let info = super.getMovieInfo();

  // Add onto the return value
  return `${info}\nPrice: ${this.price}`;
}
```

Because this method has the same name as the method in the **Movie** class, when you create an instance of the **RentalMovie** class, this method is called first. Use the **super** keyword to invoke the getMovieInfo() method defined in the **Movie** class. The return value from that method is placed into a variable named **info**. Add a line feed using '\n' followed by the **price** property value to the **info** variable and return this new string from the getMovieInfo() method. Remove the line of code **console.log(`Rental Price: ${movie.price}`);** so your code looks just like the following:

```
// Create an instance of a RentalMovie class
let movie = new RentalMovie(1,
  "Young Frankenstein", "Comedy",
  "Gene Wilder,Marty Feldman", 3.99);

// Display the movie data
console.log(movie.getMovieInfo());
```

Try It Out

Save the changes to the **class.html** file, display the page in your browser, and you should see the results from getMovieInfo() method appear in the console window including the price information.

Extend the JavaScript Number Class

Not only can you extend your classes, but you may also extend the classes pre-defined in JavaScript such as the **Number** class. Create a new file in your project folder named **number.html** and add the code shown in **Listing 3**. Within the <script> tag, add the following code snippet to extend the **Number** class.

```
class Currency extends Number {
  asUSCurrency() {
    return this.valueOf()
      .toLocaleString("en-US",
      { "style": "currency",
        "currency": "USD" });
  }
}
```

The code above gets all the functionality of the **Number** class and adds on a new method named asUSCurrency(). Use the **this.valueOf**() method to retrieve the current value in the Number object and pass that value to the toLocaleString() method. This method converts the numeric value into a U.S. currency-formatted string and returns that string value. Just below the definition of the **Currency** class, add a line of code to create a new instance of the Currency class and pass into the constructor the value 3.99, as shown in the following code:

```
let price = new Currency(3.99);
```

Write a line of code to display the value 3.99 as a U.S. currency-formatted string by calling the new asUSCurrency() method, as shown in the following line of code.

```
console.log(price.asUSCurrency());
```

Try It Out

Save the changes you made to the **number.html** file and display this file in your browser. Open the F12 developer tools and, in the console window, you should see the number displayed as **$3.99**.

Create Private Class Fields

A class does not necessarily need to expose all properties and/or methods outside of its scope. You might need some private variables to hold internal data, and maybe a method to perform some calculations. But this data and functionality does not need to be called from any code that instantiates the class. This is where a private field comes into play. Create a private field value by prefixing

the name of the field with a pound/hash sign (#). Open the **Movie.js** file and, just after the constructor, add a private field as shown below:

```
#releaseDate;
```

Within the constructor, initialize this private field as shown in the following code snippet:

```
constructor(id, name, genres, actors) {
  // Initialize public instance properties
  this.filmId = id;
  this.name = name;
  this.genres = genres;
  this.actors = actors;

  // Initialize private fields
  this.#releaseDate = "2024-01-01";
```

In the getMovieInfo() method, return the private field value in the template literal string. This code is split onto multiple lines for the purposes of formatting in this printed magazine. Be sure to place all this code on a single line when entering it into your program.

```
getMovieInfo() {
  return `Film ID: ${this.filmId}\n
     Name: ${this.name}\n
     Genres: ${this.genres}\n
     Actors: ${this.actors}\n
     Release Date: ${this.#releaseDate}`;
}
```

Open the **class.html** file and add the following code at the end of the other code to see that **undefined** is displayed in the console window:

```
console.log(movie.releaseDate);
```

Try It Out

Display the **class.html** file in your browser, open the F12 Developer Tools, and the console window should look like **Figure 3**. Notice the **releaseDate** variable is displayed

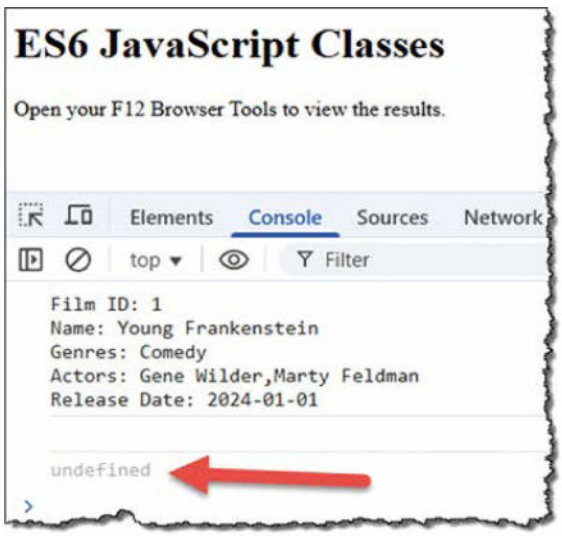


Figure 3: If you try to access a private field in a class, an undefined is returned instead.

dtSearch®

Instantly Search Terabytes

Enterprise and developer products have:

- over 25 different search features
- credit card search and other forensics-oriented options
- efficient multithreaded indexing and searching

dtSearch's document filters support:

- local and remote "Office" files, PDFs, compression formats, etc.
- emails with multilevel attachments
- a wide variety of databases
- web data

Developers:

- SDKs for Win / Linux / macOS
- APIs for current .NET / C++ / Java
- faceted search, granular data classification and other API options
- deploy on-premises or in cloud

Visit dtSearch.com

- **for hundreds of reviews and case studies**
- **for fully-functional enterprise and developer evaluations**

The Smart Choice for Text Retrieval® since 1991

dtSearch.com | 1-800-IT-FINDS

from the getMovieInfo() method. However, when you attempt to access it from the movie object, the result is undefined.

Use Getters and Setters to Create Public Properties

In addition to holding internal data for a class, private class fields are frequently used to expose data from a getter or a setter function. The keywords **get** and **set** used before a name in a class create a public property from a private class field. Open the **Movie.js** file and add the code shown in the following snippet:

```
set releaseDate(value) {
  this.#releaseDate = value;
}

get releaseDate() {
  return this.#releaseDate;
}
```

When a **set** function is created, you must have a **value** passed into the function. This allows you to place a value on the right-hand side of an equal sign to pass the value into the **set** function, as shown in the following code:

```
movie.releaseDate = "1974-01-01";
```

When this code executes, the value of "1974-01-01" is passed to the **value** parameter in the **set releaseDate(value)** function. This in turn is placed into the **#releaseDate** private field. When you wish to query the **releaseDate** property through the movie object, the **get** function is called. For example, the following line of code calls the releaseDate **get** function and returns the current value of the **#releaseDate** private field and passes it to the console.log() function.

```
console.log(movie.releaseDate);
```

As you have now guessed, all this code is equivalent to just creating a public property on the class. So, why would you use the getter and setter functions? There are a few reasons. First, you may only wish to create a write-only property, in which case you only create a setter function to set the value for a private field. Second, you may wish to create a read-only property, in which case, you only create a getter function to return the value for the private field. Third, you may wish to add some additional code within the get or set functions to manipulate the private field prior to getting or setting the value.

Create Private Class Methods

In the **scripts** folder, create a new file named **Product.js** and add a new class definition, as shown in **Listing 4**. The Product class has three public properties: **productId**, **name**, and **profit**. It also contains two private fields named **#cost** and **#price**. There are corresponding property getter and setter functions to get and set the #cost and #price fields.

Add a private method to the Product class to calculate and set the **profit** property whenever the #cost or #price fields are modified. A private method is created just like you create private fields: Add a pound/hash (#) sign before the name of the function. Add a new private method to the Product class named **#calculateProfit()**, as shown in the following code.

```
#calculateProfit() {
  console.log("In the calculateProfit() method");

  // Ensure you have a valid expression by
  // using the nullish coalescing operator
  this.profit = (this.price ?? 0) -
    (this.cost ?? 0);
}
```

Once you have added the #calculateProfit() method, modify each property setter to call this method after setting each of the respective private fields, as shown in the code below:

```
set cost(value) {
  this.#cost = value;
```

Listing 4: Create a Product class that automatically calculates the profit of a product using a private method.

```
class Product {
  // Public Property Definitions
  productId;
  name;
  profit = 0;

  // Private Field Definitions
  #cost;
  #price;

  // Constructor Definition
  constructor(id, name, cost, price) {
    // Initialize public instance properties
    this.productId = id;
    this.name = name;

    this.#cost = cost;
    this.#price = price;
  }

  // Public Property Getters and Setters
  get cost() {
    return this.#cost;
  }
  set cost(value) {
    this.#cost = value;
  }

  get price() {
    return this.#price;
  }
  set price(value) {
    this.#price = value;
  }

  // Public Method Definitions
  getProductInfo() {
    return `Product ID: ${this.productId}\n
      Name: ${this.name}\n
      Cost: ${this.cost}\n
      Price: ${this.price}\n
      Profit: ${this.profit}`;
  }
}
```

Listing 5: Create a product page to display the Product class information.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">

  <title>Private Method</title>
</head>

<body>
  <header>
    <h1>Private Method</h1>
  </header>

  <main>
    <p>Open your F12 Browser Tools
       to view the results.</p>
  </main>

  <script src="classes/Product.js">
  </script>

  <script>
    'use strict';

    let prod = new Product(1,
      "Acme Bazooka", 100, 300);

    // Display the product data
    console.log(prod.getProductInfo());
    console.log("");

    console.log("Setting cost to 200
      to force recalculation of profit.");
    prod.cost = "200";

    // Display the product data
    console.log(prod.getProductInfo());
  </script>
</body>

</html>
```

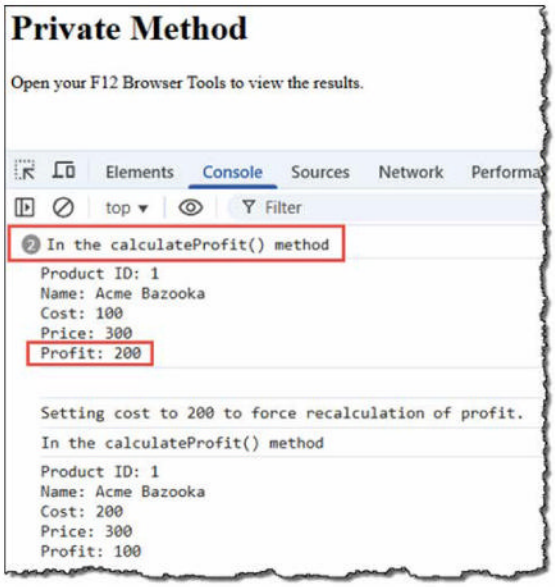


Figure 4: Create a private method to calculate the profit of a product.

```
  // Re-calculate the profit
  this.#calculateProfit();
}

set price(value) {
  this.#price = value;

  // Re-calculate the profit
  this.#calculateProfit();
}
```

Within the constructor of the Product class, call the setter for both the cost and price fields so the #calculateProfit() method is called when the values for the cost and price fields are passed in. Make sure you don't set the private fields #cost and #price in the constructor because then the profit is not calculated.

Make the change to the constructor as shown in the code snippet below. Because you don't know what someone might pass into the constructor for the cost and price, it's a good idea to use the nullish coalescing operator to ensure that any **null** or **undefined** value passed in is converted to a zero.

```
constructor(id, name, cost, price) {
  this.productId = id;
  this.name = name;

  // Call the setters to ensure
  // the profit is calculated
  this.cost = cost ?? 0;
  this.price = price ?? 0;
}
```

Create a new file in the root folder of your project named **product.html** and add the code shown in **Listing 5**.

Try It Out

Save all the changes you have made and display the **product.html** file in your browser. Open the F12 Developer Tools, and you should see a page that looks like **Figure 4**.

Static Properties and Methods

In JavaScript, use the **static** keyword to create static properties and methods. Static properties are used to hold data in a class that is then available through the class name. Only a single instance of this data is created the first time the class is loaded. Static methods are used for utility functions that do not rely on data in an instance of a class. Examples of static methods are those used to clone or create an instance of an object of the class. When you use static properties or methods, you access them only through the class name itself, and not through an instance variable of the class.

There are several examples of static methods in the JavaScript language itself, for example, Math.max(), Date.Now(), JSON.parse(), and JSON.stringify(). All these static methods are called on the class name itself and may or may not have arguments passed to them. Each of these methods performs an operation and returns a value.

There are several examples of static properties in the JavaScript language too. For example, Number.MIN_VALUE,

Number.MAX_VALUE, and Math.PI. These are all read-only properties, but they are static because they are only accessed from the Number class, not from a number object.

Uses of Static Properties and Methods

Just as the JavaScript language uses static properties and methods, you will find multiple uses for them in your applications as well. Some property examples might be to store cached data that can be shared across all instances of a class. Define constants that are used to represent a fixed value for classes, such as an index number, into an array of pre-defined values. Another example might be a set of properties that represent configuration values needed in your application.

Static functions are utility or helper functions where you pass in an argument(s) and it returns some value. A good example of a static method would be one to make an API call to retrieve some data, or a database call to modify some data. Static methods are often used to implement a factory pattern, where a class is responsible for creating objects of other classes. Another good use of static methods is to create a clone of an existing object of the class in which the static method is declared.

Static Class Properties

To illustrate class properties, add two static properties to the Movie class. The first property is **counter** and is used to keep track of how many instances of the Movie class have been created. The second property is **initDate**, which is the date and time the Movie class is loaded by JavaScript. Open the **Movie.js** file and add the code shown in the following snippet:

```
static counter = 0;
static initDate = new Date();
```

Add a constructor to increment the counter each time a new instance of a Movie object is created. Notice that you are using the class name and not the keyword **this** because counter is a static property.

```
constructor() {
  // Increment # of Movie objects created
  ++Movie.counter;
}
```

Within the <script> element on the page, add the following code to create an instance of a Movie class. Try to access the counter property through the instance variable and you will see a value of **undefined** returned. Static properties cannot be accessed through an instance variable, so write code to use the Movie class name followed by the static properties to display the current static property values.

```
console.log("Creating a new Movie instance");

// Create an instance of a Movie class
let movie = new Movie();

// Can't access static properties
// through an instance
console.log(`movie.counter = ${movie.counter}`);

// Access static properties using class name only
console.log(`Movie.counter = ${Movie.counter}`);
console.log(`Movie.initDate = ${Movie.initDate}`);
console.log("");
```

Try It Out

Display the **class.html** file in your browser. Open your F12 Developer Tools in your browser and you should see the values undefined, zero, and the current date and time.

Create Another Instance of a Movie Class

Add additional code after the code you just ran to create another instance of a Movie class. After creating this new instance, display the counter static property again and you should see that the static counter property has incremented by one.

```
let movie2 = new Movie();
console.log("Creating another Movie instance");
console.log(`Movie.counter = ${Movie.counter}`);
console.log("");
```

Try It Out

Display the **class.html** file in your browser. Open your F12 Developer Tools in your browser and, after the console values displayed in the previous sample, you should now see that the counter static property is equal to the number two (2).

Manually Increment the Counter

Static properties can be manipulated just like instance properties. Write the code shown below to increment the counter property.

```
// Static public properties are read/write
console.log("Increment the Movie counter");
Movie.counter++;
```

Write code to display the counter and initDate properties after waiting for approximately two seconds. The reason for delaying the running of this code is to illustrate that the date/time has not changed since the Movie class was first loaded by JavaScript. Static properties contain a single value across all instances and do not change unless specifically changed by your code.

```
// This code runs after 2 seconds to show
// the 'initDate' static property is the same
window.setTimeout(() => {
  // Display the static movie data
  console.log(`Movie.counter = ${Movie.counter}`);
  console.log(`Movie.initDate =
    ${Movie.initDate}`);
}, 2000);
```

Try It Out

Display the **class.html** file in your browser. Open your F12 Developer Tools in your browser and after the console values displayed in the previous two samples, you should now see that the counter static property is equal to the number three (3) and the initDate property has not changed between the time the script was loaded and when this code was run, which is two seconds after loading the class.

Getting the Sample Code

You can download the sample code for this article by visiting www.CODEMag.com under the issue and article, or by visiting www.pdsa.com/downloads. Select "Articles" from the Category drop-down. Then select "Add Some Class to Your JavaScript" from the Item drop-down.

Static Read-Only Class Properties

Open the **Movie.js** file and change the static properties you created to be private static fields by prefixing the names with the pound/hash (#) sign. Modify the constructor to increment the #counter field. Add two static **get** functions to return the two static private fields.

```
static #counter = 0;
static #initDate = new Date();

constructor() {
  // Increment # of Movie objects created
  ++Movie.#counter;
}

static get counter() {
  return this.#counter;
}

static get initDate() {
  return this.#initDate;
}
```

Replace all the code within the <script> element with the following snippet. In this snippet, you are displaying the two static read-only properties. The last line attempts to set the initDate property, but because this property only has a getter, this line causes an exception to be displayed in the console window.

```
// Access static properties on the class name only
console.log(`Movie.counter = ${Movie.counter}`);
console.log(`Movie.initDate = ${Movie.initDate}`);
console.log("");

// Can't write to read-only static properties
// NOTE: The following line causes an
//       exception in the console window
Movie.initDate = new Date();
```

Try It Out

Display the **class.html** file in your browser. Open your F12 Developer Tools in your browser and you should see that the counter is zero (0) and the initDate is the current date/time, and then you should see an exception where you attempted to set the initDate read-only property. You can create write-only static properties as well, but in practice, I doubt you will find much use for them.

Static Constructors

A static constructor is a special function that runs one time when the class is loaded. Use a static constructor to initialize any static properties to a valid starting value. Open the **Movie.js** file and add a static constructor using the **static** keyword followed by an open curly brace. Place code in between this opening curly brace and a closing curly brace. In the following code, initialize the static properties counter and initDate by using the **this** keyword. This seems counterintuitive as the **this** keyword is generally used for instance properties in a class, but because the code is running in the static constructor, JavaScript knows to use **this** to access the static properties.

```
// Static Constructor
static {
  console.log("In the static constructor");

  // Initialize a static property using 'this'
  this.#counter = 0;
  this.#initDate = new Date();
}
```

Modify the code within the <script> element to look like the following snippet.

```
console.log("Script is starting");

// Access static properties on the class name only
console.log(`Movie.counter = ${Movie.counter}`);
console.log(`Movie.initDate = ${Movie.initDate}`);
console.log("");
```

Try It Out

Display the **class.html** file in your browser. Open your F12 Developer Tools in your browser and you should see the message "In the static constructor" appear first, then the message "Script is starting", in that order. This illustrates that the static constructor is run immediately when the page is loaded and before any code in the <script> element is run.

Static Class Methods

As mentioned previously, static methods can be created within a JavaScript class. To create a static method, prepend the **static** keyword before the name of the function. Open the **Movie.js** file and add the following static method to create a clone of an existing Movie object. This method accepts a parameter that is a movie object. It then creates a new instance of a Movie object and uses the Object.assign() method to copy all property values from the passed in movie object into the new instance. The cloned Movie object is then returned from this method.

```
// Static method to create a Movie instance
static clone(movie) {
  let clone = new Movie();

  // Clone the original data
  Object.assign(clone, movie);

  return clone;
}
```

Delete all the code within the <script> element. Add code to create an instance of a Movie object and call the getMovieInfo() method to display the movie information, as shown in the following code snippet:

```
// Create an instance of a Movie class
let movie = new Movie(1,
  "Young Frankenstein", "Comedy",
  "Gene Wilder,Marty Feldman");

// Display the movie data
console.log(movie.getMovieInfo());
console.log("");
```

Add code to create a clone of the movie object by calling the clone() method passing in the **movie** object variable.

SPONSORED SIDEBAR:

Ready to Modernize a Legacy App?

Need advice on migrating yesterday's **legacy applications** to today's modern platforms?

Take advantage of **CODE Consulting's** years of experience and contact us today to schedule a **FREE** consulting call to discuss your options.

No strings.
No commitment.

For more information, go to www.codemag.com/consulting or email us at info@codemag.com.

Listing 6: Create code to test the static properties and methods of the MathHelper class.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">

  <title>Math Helper Class</title>
</head>

<body>
  <header>
    <h1>Math Helper Class</h1>
  </header>

  <main>
    <p>Open your F12 Browser Tools
       to view the results.</p>
  </main>

  <script src="classes/MathHelper.js"></script>
  <script>
    'use strict';

    console.log(`PI = ${MathHelper.PI}`);
    console.log("");

    console.log(`MathHelper.add(42, 42) =
      ${MathHelper.add(42, 42)}`);
    console.log(`MathHelper.multiply(42, 42) =
      ${MathHelper.multiply(42, 42)}`);
  </script>
</body>

</html>
```

Assign the return value from the clone() method into a variable named **clone**. Change the **name** property on the **clone** object and call the getMovieInfo() method to display the movie information with the new movie name. Finally, call the getMovieInfo() method one more time on the original movie object to show that the two objects are distinct from one another.

```
// Clone the original object
let clone = Movie.clone(movie);
clone.name = "Changed Movie Name";
// Display the new movie data
console.log(clone.getMovieInfo());
console.log("");

// Display the original movie data
console.log(movie.getMovieInfo());
```

Try It Out

Display the **class.html** file in your browser. Open your F12 Developer Tools in your browser and you should see the different movie information in the console window.

Math Helper Class

A good use of static properties and methods is to create a class with nothing but utility helper properties and methods. To illustrate, create a folder named **classes** and add a new file named **MathHelper.js**. Add the following code into this new file:

```
class MathHelper {
  static get PI() {
    return 3.14159;
  }

  static add(op1, op2) {
    return op1 + op2;
  }

  static multiply(op1, op2) {
    return op1 * op2;
  }
}
```

The MathHelper class contains a single read-only property named PI that returns the value of PI to the fifth most significant digit. Two methods add() and multiply() are also created. Although you don't really need to create these methods because addition and multiplication are easily done in JavaScript, I wanted to show an example of a class with nothing but static properties and methods. To test the MathHelper class, create a file named **math-helper.html** and add the HTML shown in **Listing 6** into the new file.

Try It Out

Display the **math-helper.html** file in your browser. Open your F12 Developer Tools in your browser and you should see the value of PI displayed in the console window along with the results from calling the add() and multiply methods.

Application Settings Class

Another time you might need to use static in a class is for application-wide application settings. In all but the most

ADVERTISERS INDEX

JavaScript, C#, Angular, Machine Learning, AI, Security
CODE
Integrating AI and ML in .NET Core with ML.NET
Add Custom Scripting to Your Apps
Understand Model Context Protocol (MCP)
JavaScript Classes 101

Advertising Sales:
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

Advertisers Index

BrickCon	www.brickcon.ai	5
CODE Consulting--AI Services	www.codemag.com/ai-services	2
CODE Consulting--Executive Briefing	www.codemag.com/executivebriefing	7
CODE Consulting--Copilot	www.codemag.com/copilot	31
CODE Consulting	www.codemag.com/Code	75
CODE Legacy	www.codemag.com/legacy	55
CODE Staffing	www.codemag.com/staffing	76
dtSearch	www.dtSearch.com	13
Geo Week	www.geo-week.com	23

This listing is provided as a courtesy to our readers and advertisers. The publisher assumes no responsibility for errors or omissions.

Listing 7: Create an application settings class to hold global values for your application.

```
class ApplicationSettings {
  // Private Static Property Definitions
  static #apiUrl;
  static #port;

  // Public Static Property Definitions
  static get apiUrl() {
    return this.#apiUrl;
  }

  static get port() {
    return this.#port;
  }

  // Public Method Definitions
  static load() {
    // TODO: Load these from a configuration file
    this.#apiUrl = "https://www.pdsa.com/api";
    this.#port = "8080";
  }
}
```

Listing 8: Add an HTML page to test the ApplicationSettings class.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">

  <title>Application Settings Class</title>
</head>

<body>
  <header>
    <h1>Application Settings Class</h1>
  </header>

  <main>
    <p>Open your F12 Browser Tools to
       view the results.</p>
  </main>

  <script src="classes/ApplicationSettings.js">
  </script>
  <script>
    'use strict';

    // Load application settings
    ApplicationSettings.load();

    console.log(`ApplicationSettings.apiUrl
      = ${ApplicationSettings.apiUrl}`);
    console.log(`ApplicationSettings.port
      = ${ApplicationSettings.port}`);
  </script>
</body>

</html>
```

simple of applications, you most likely have several global values that you wish to use among all your different HTML pages. Examples of global values include, the URL used to access API endpoints, the port number to use, a connection string to connect to a SQL database, or the name of a database when using IndexedDB. Add a new file named **ApplicationSettings.js** within the **classes** folder and add the code shown in **Listing 7**.

To test the ApplicationSettings class, create a file named **application-settings.html** and add the HTML shown in **Listing 8**. Within the <script> element, you first make a call to the static method named load() to initialize the **apiUrl** and **port** properties. These values are hard coded in this simple sample, but you may wish to read them from a configuration file in your application.

Try It Out

Display the **application-settings.html** file in your browser to see the values contained in each of the static properties.

Summary

In this article, you were introduced to using classes in JavaScript. You learned to define classes with properties and methods. You saw a few different methods of instantiating a class and learned that classes are not hoisted and must be defined before any code that uses the class. There are many options for creating and working with properties of a class. You have the option of automatic properties, declaring properties explicitly, creating read-only, and write-only properties. Inheritance is one of the best features of classes and you can take full advantage of this feature including overriding methods. I hope you start taking advantage of JavaScript classes in your applications.

Paul D. Sheriff

Threat Detection Using Anomaly Detection

I had an interesting conversation on Twitter/X recently. It was about security in the world of AI. You can check it out here https://x.com/sahilmalik/status/1960549394425520621. This is, in fact, something that I've been thinking about for a long time. I've been a developer for a few decades now, and as I've seen productivity increasing using greater and greater layers of abstraction, none of them scare me as much as artificial intelligence.

Sahil Malik
www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

His areas of expertise are cross-platform Mobile app development, Microsoft anything, and security and identity.

We are almost turning from developers into psychologists, where we give computers instructions in plain English, and the computer generates a lot of code for us. How good is this code really? There are some really smart people on the other side that are looking for the tiniest vulnerability in your code that you may not have even thought of.

I'm used to a world where developers spend a lot of time polishing every bit of their code, even the ones you don't see, and make it nearly perfect so there are no security holes. And yet hackers smarter than them end up finding those holes.

How is this picture going to change in a world of artificial intelligence where developers are using artificial intelligence to generate a lot of code? Increasingly, it's becoming more and more clear to me that the way to fight the security battle in this artificial intelligence world is with artificial intelligence.

In practice, I don't see this happening. Security always seems to be an afterthought, and productivity seems to be the forethought. Security is important, and in this AI world, it's even more important. So, I thought I would write an article on this topic.

> Productivity seems to be a priority with security as an afterthought.

This landscape of security using artificial intelligence, or, frankly, security as it applies to artificial intelligence, is a vast topic in itself. I always strive to give you working examples, something you can try right away, and that will be my focus in this article as well.

In this article, I'm going to walk you through building a simple, entirely local anomaly detection system using Python. You'll simulate system log data, train an Isolation Forest model, and use it to identify suspicious activities. The best part? Everything runs right on your Mac, no cloud services required. Of course, as is the theme of many of my articles, you can leverage cloud-based services to greatly enhance these capabilities. It will be a piece of code that works 100% locally, has isolation and control advantages, and, frankly, sometimes, cost advantages.

Everything I'm about to show in this article will run on your MacOS. You could also make things work on a Windows machine, but I developed this codebase on a Mac. You'll need Python 3 and a relatively beefy modern processor. That's about it.

With that, let's get started.

Why Isolation Forest for Anomaly Detection?

I choose Isolation Forest for anomaly detection for several reasons.

First, it's extremely efficient, even with large datasets. It's efficient primarily because of its linear time complexity and low memory requirements. This is essential because I intend to run everything locally.

Secondly, Isolation Forest is particularly effective for high-dimensional data because it doesn't rely on distance or density metrics to detect anomalies. Many traditional anomaly detection algorithms, like k-Nearest Neighbors or Local Outlier Factor (LOF), rely on calculating distances between data points. In high-dimensional spaces, this approach suffers from the curse of dimensionality, where the distance between all data points becomes nearly uniform, making it impossible to identify meaningful neighbors or density variations. Isolation Forest bypasses this problem by using a tree-based approach.

Finally, Isolation Forest doesn't require labelling data. It's an unsupervised learning algorithm, meaning it doesn't require pre-labeled "normal" or "malicious" data to learn. It simply identifies data points that are "different." You can literally throw the code I'm going to show in this article at any dataset and it will find anomalies for you.

Setting Up Your Environment

In order to follow along in this article, I expect that you are at least a basic Python developer. I'm not going to explain basic things like PIP or installing PIP packages. I'm going to assume familiarity with an IDE like VS Code.

With that out of the way, set up a Python project for yourself and open it in VS code. Create a virtual environment in it as well.

In this virtual environment, you're going to need certain dependencies. The contents of my requirements.txt can be seen as below:

```
pandas
scikit-learn
numpy
matplotlib
seaborn
```

With your requirements.txt saved, install your requirements using the command below:

```
pip install -r requirements.txt
```

Excellent, now you're going to need some test data.

Creating Test Data

The code I'm about to show will work on any input data. One of the challenges about writing things in the form of a published article is figuring out what data can I show you that's somewhat real world and also doesn't intrude on anybody's intellectual property.

For that reason, I decided to mimic a real-world scenario using some sample data that I'm going to generate next. In the real world, you'd ingest actual system logs, such as authentication logs, network traffic, process data, etc.

For our purposes, let's generate some synthetic data that mimics normal user activity and let's inject some anomalies for fun.

My sample log file contains the following columns:

- login_attempts_per_min: Number of login attempts
- files_accessed_per_min: Number of unique files accessed
- cpu_usage_avg: Average CPU usage
- network_out_mb: MB of data sent over the network
- process_count: Number of active processes

Create a file called **generate_data.py** and let's start writing some code to generate this data. To start with, add some imports at the top of this file, as shown below:

```
import pandas as pd
import numpy as np
import random
```

The pandas library is a powerful tool for working with tables of data, which are called DataFrames. The NumPy library is used for advanced math and for creating arrays of numbers, which is great for generating large amounts of data. The random library helps with things like shuffling the data.

Next let's create some normal data. The code for generating normal data can be seen in **Listing 1**. As you can see from **Listing 1**, the generate_normal_data function acts like a factory for typical system activity. It creates a dictionary (a set of key-value pairs) where each key is a feature of system behavior, like login_attempts_per_min or cpu_usage_avg. The values are lists of random numbers generated by NumPy that fall within a normal, expected range. For example, it creates 1,000 data points where the cpu_usage_avg is a random number between 5% and 40%. The function then converts this dictionary into a pandas DataFrame, which is essentially a spreadsheet-like table.

Now, in a similar vein, let's write up a method to generate anomalous data. This can be seen in **Listing 2**. As you can see from **Listing 2**, The generate_anomalous_data function is similar to the generate normal data function, but it creates data that's unusual or suspicious. It generates a smaller number of data points (20 by default) where the values are outside the normal range. For instance, cpu_usage_avg is set to a high value between 80% and 100%, and network_out_mb is also very high. These high values simulate a potential security breach or a system error, making them anomalies. This is crucial for training an anomaly detection model later.

Listing 1: Normal data generation code

```
def generate_normal_data(num_samples=1000):
    """Generates synthetic 'normal' system activity data."""
    data = {
        'login_attempts_per_min':
            np.random.randint(0, 3, num_samples),
        'files_accessed_per_min':
            np.random.randint(1, 20, num_samples),
        'cpu_usage_avg':
            np.random.uniform(5, 40, num_samples),
        'network_out_mb':
            np.random.uniform(0.1, 5.0, num_samples),
        'process_count':
            np.random.randint(30, 80, num_samples)
    }
    return pd.DataFrame(data)
```

Listing 2: Anomalous data generation code

```
def generate_anomalous_data(num_anomalies=20):
    """Generates synthetic 'anomalous' system activity data."""
    anomalies = {
        # Many failed logins
        'login_attempts_per_min':
            np.random.randint(10, 50, num_anomalies),
        # Accessing many files
        'files_accessed_per_min':
            np.random.randint(50, 200, num_anomalies),
        # High CPU
        'cpu_usage_avg':
            np.random.uniform(80, 100, num_anomalies),
        # High network egress
        'network_out_mb':
            np.random.uniform(50.0, 200.0, num_anomalies),
        # Many processes
        'process_count':
            np.random.randint(100, 200, num_anomalies)
    }
    return pd.DataFrame(anomalies)
```

Listing 3: Generating, mixing up, and saving data

```
if __name__ == "__main__":
    normal_df = generate_normal_data()
    anomalous_df = generate_anomalous_data()

    # Combine and shuffle for a realistic dataset
    combined_df = pd.concat(
        [normal_df, anomalous_df],
            ignore_index=True)
    combined_df = combined_df.sample(
        frac=1).reset_index(
            drop=True) # Shuffle rows

    # Save to CSV
    combined_df.to_csv(
        "system_activity_logs.csv", index=False)
    print(f"Generated
      {len(normal_df)} normal samples
      and {len(anomalous_df)} anomalies.")
    print("Data saved to system_activity_logs.csv")
    print("\nFirst 5 rows of generated data:")
    print(combined_df.head())
```

```
✘ sahilmalik@Makhi ~/Documents/Developer/cm/anomaly python ./generate_data.py
Generated 1000 normal samples and 20 anomalies.
Data saved to system_activity_logs.csv

First 5 rows of generated data:
   login_attempts_per_min  files_accessed_per_min  cpu_usage_avg  network_out_mb  process_count
0                       0                      13       7.317640        4.701002             74
1                       1                       3      26.231286        2.379521             41
2                       2                      18      34.292955        3.081814             54
3                       1                       8      22.614534        0.932157             79
4                       0                      15       5.225384        0.989876             44
```

Figure 1: Sample data is ready to go

SPONSORED SIDEBAR:

CODE Is Hiring!

CODE Staffing is accepting resumes for various open positions ranging from junior to senior roles.

We have **multiple openings** and will consider candidates who seek full-time employment or contracting opportunities.

For more information
www.codestaffing.com.

Now that you have functions to generate normal and anomalous data, let's use the **__main__** function, to generate data, and mix them up. This can be seen in **Listing 3.** As you can see from **Listing 3**, the __main__ method first calls the two functions created earlier to get a table of normal data and a table of anomalous data. It then uses pd.concat to stack the two tables on top of each other, creating one large table. The combined table is then shuffled randomly using .sample(frac=1) so that the normal and anomalous data are mixed together, making the dataset more realistic.

Finally, combined_df.to_csv("system_activity_logs.csv", index=False) saves the complete dataset into a file named system_activity_logs.csv. The index=False part just means that it doesn't add an extra column with row numbers to the file.

Run this file using the following command:

```
python ./generate_data.py
```

If all is correct so far, you should see some random data generated in system_activity_logs.csv file, and sample output, as can be seen in **Figure 1**.

Building the Anomaly Detection System

Nicely done so far, if you're following along. Now that you have some sample data ready to go let's write a simple anomaly detection system. As I mentioned earlier, you're going to write some code that uses a machine learning algorithm called Isolation Forest to find anomalies (unusual data points) in a dataset of system activity logs. The main goal is to spot suspicious activities, like a sudden spike in CPU usage or network traffic, that don't fit the normal pattern.

There will be four main parts to this code. The first part is going to load the data. The second part is going to train the model. Then you'll predict anomalies. Finally, you'll visualize the results. And all this will be coordinated with a __main__ method.

Create a new file called "detect_anomalies.py".

Listing 4: Traini the model

```
def train_model(df, contamination=CONTAMINATION_RATE):
    print("Training Isolation Forest model...")
    model = IsolationForest(
       contamination=contamination, random_state=42)
    model.fit(df)
    print("Model training complete.")
    return model
```

First, let's get the imports out of the way.

```
import pandas as pd
import numpy as np
from sklearn.ensemble
  import IsolationForest
import matplotlib.pyplot as plt
import seaborn as sns
```

The two new libraries you see here are Matplotlib and seaborn. You're going to use them to perform data visualization. The primary purpose is to create informative and visually appealing statistical graphics, simplifying the process of exploring and presenting data.

Next, let's define some constants at the top.

```
DATA_FILE = "system_activity_logs.csv"
 # For saving/loading the model
MODEL_PATH = "isolation_forest_model.pkl"
# Expected proportion of anomalies in the
# dataset (2%)
CONTAMINATION_RATE = 0.02
```

Now let's perform the first step, which is to load the data, and which can be seen here:

```
df = pd.read_csv(DATA_FILE)
```

Okay that was simple enough. Let's move on to the second step where you train the model. The code training the model can be seen in **Listing 4.** As can be seen in **Listing 4**, the train_model function creates and trains the Isolation Forest model. Think of this model as a smart detective that learns what's considered normal behavior based on the data you provide. The contamination setting tells the detective to expect a certain percentage (in this case, 2%) of the data to be anomalies, which helps it set a good baseline for what to look for. You can tweak this parameter to make your model less or more sensitive to anomalies.

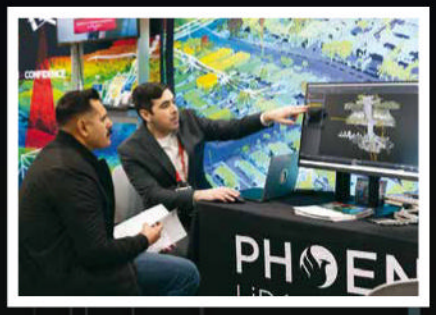
Now that the model is trained, the next step is to use it to predict anomalies. This can be seen in **Listing 5**. As you can see from **Listing 5**, the predict_anomalies function uses the trained model to go through the data and make a prediction for each data point. For each activity log, the model assigns an anomaly score, which measures how "weird" or isolated that data point is. It then gives a final label: **-1** for an anomaly and **1** for a normal data point.

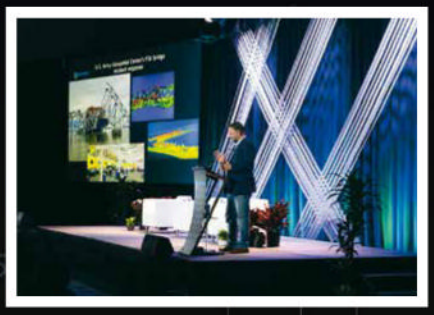
This seems almost too simple, doesn't it? Now let's add some visualization to this data, so it looks like I did some-

GEO WEEK

FEBRUARY 16-18, 2026 | DENVER, CO - USA
geo-week.com

HARNESS THE POWER of GEOSPATIAL





Accomplish a year's worth of geospatial business in just three days.

Geo Week brings together geospatial and mapping professionals to explore how data, maps, and advanced technologies can deepen our understanding of the world and inform solutions for complex challenges.

Geo Week's conference program and tradeshow floor provide expert insights, real-world case studies, cutting edge tools, and unmatched peer-to-peer-networking. **Harness the power of geospatial at Geo Week!**

REGISTER AT GEO-WEEK.COM
Use code **SAVE100** for $100 off a conference pass or a **FREE** exhibit hall pass.

Produced by diversified

PRESENTED BY: GEO WEEK NEWS

EVENT PARTNERS: asprs THE IMAGING & GEOSPATIAL INFORMATION SOCIETY | RCN

INDUSTRIES SERVED

- Surveying & Mapping
- Architecture, Engineering & Construction
- Infrastructure & Transportation
- Energy, Utilities, Telecom & Industrial
- Natural Resource Management
- Historical Preservation & Documentation
- Facility Management
- Urban Planning & Development
- Disaster & Emergency Response

thing smart and intelligent in this article. As you can see in **Listing 6**, the code prints out the number of anomalies found and lists some of the most unusual ones. The visualize_results function then creates two helpful graphs.

The first is a scatter plot that plots two key features (e.g., CPU usage vs. network traffic) and colors the unusual points red so you can easily see them.

Listing 5: Predict anomalies

```
def predict_anomalies(model, df):
    """Predicts anomalies using the trained model."""
    print("Predicting anomalies...")

    # Create a copy of the DataFrame to add new columns to
    df_with_predictions = df.copy()

    # IF returns -1 for anomalies and 1 for normal
    df_with_predictions['anomaly_score'] =
       model.decision_function(df)
    df_with_predictions['is_anomaly'] =
       model.predict(df)

    return df_with_predictions
```

Listing 6: Visualizing the data

```
def visualize_results(df):
    print("Generating visualizations...")

    plt.figure(figsize=(16, 6))

    # Scatter plot of two features, colored by anomaly status
    plt.subplot(1, 2, 1)
    sns.scatterplot(
        x='cpu_usage_avg',
        y='network_out_mb',
        hue='is_anomaly',
        palette={1: 'green', -1: 'red'},
        data=df, s=100, alpha=0.7)
    plt.title('CPU Usage vs. Network Out(Anomalies in Red)')
    plt.xlabel('Average CPU Usage (%)')
    plt.ylabel('Network Out (MB)')

    # Distribution of Anomaly Scores
    plt.subplot(1, 2, 2)
    sns.histplot(df['anomaly_score'], kde=True, bins=50)
    plt.axvline(
        x=df[df['is_anomaly'] == -1]['anomaly_score'].max(),
        color='red',
        linestyle='--',
        label='Anomaly Threshold')
    plt.title('Distribution of Anomaly Scores')
    plt.xlabel('Anomaly Score')
    plt.ylabel('Count')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

The second is the anomaly score distribution that shows how many data points have each score. You can see a clear line that separates the few data points with very low (anomalous) scores from the large group of normal ones.

Now let's put all this together in a main() function to load the data, train the model, predict using the model, and visualize the results. This can be seen in **Listing 7**.

Run the anomaly detection code as follows:

```
python ./detect_anomalies.py
```

This should produce an output, as can be seen in **Figure 2**.

Because you added a visual to the anomalies, you can see that in action in **Figure 3**. You'll see output indicating the model training, prediction, and a summary of detected anomalies. Specifically, you can see two plots in **Figure 3**: a scatter plot showing your data points with detected anomalies highlighted in red, and a histogram of anomaly scores with a line indicating the threshold where points are considered anomalous. This visual feedback is incredibly helpful for understanding why certain points were flagged.

Finally, if you observe **Listing 7** closely, you'll also see a commented line where you could send alerts on the anomalies to an interested party.

Taking This Further

It's quite incredible that without much difficulty or code, I was able to produce an anomaly detection engine that could work against any data. Throw any logs at this data, like EntraID sign in logs, and see if it detects anomalies. You could take this much further in the real world. Next are some enhancements or tweaks you could make.

You could adjust the CONTAMINATION_RATE. The contamination parameter in Isolation Forest is crucial. It's your estimate of the proportion of anomalies in your dataset. If you set it too high, you might flag normal data as anomalous (false positives). If you set it to too low, you might miss real threats (false negatives). Experiment with this value to fine-tune your model.

You could look into feature engineering. In a real scenario, you'd spend significant time on feature engineering. Instead of raw counts, you might calculate ratios (e.g., failed_logins/total_logins), rates of change, or aggregate data over different time windows.

```
sahilmalik@Makhi ~/Documents/Developer/cm/anomaly python ./detect_anomalies.py
Loaded data from system_activity_logs.csv. Shape: (1020, 5)
Training Isolation Forest model...
Model training complete.
Predicting anomalies...

Found 21 anomalies and 999 normal data points.

--- Detected Anomalies ---
     login_attempts_per_min  files_accessed_per_min  cpu_usage_avg  network_out_mb  process_count  anomaly_score  is_anomaly
112                      41                     197      98.947636      198.200295            189      -0.266655          -1
273                      42                     153      98.204270      163.073204            142      -0.247944          -1
464                      36                     110      92.642599      141.292219            189      -0.244048          -1
500                      42                     146      85.074947      186.510790            176      -0.242961          -1
383                      18                      80      93.327280      176.604505            194      -0.238517          -1
Generating visualizations...
```

Figure 2: Anomaly detection in action

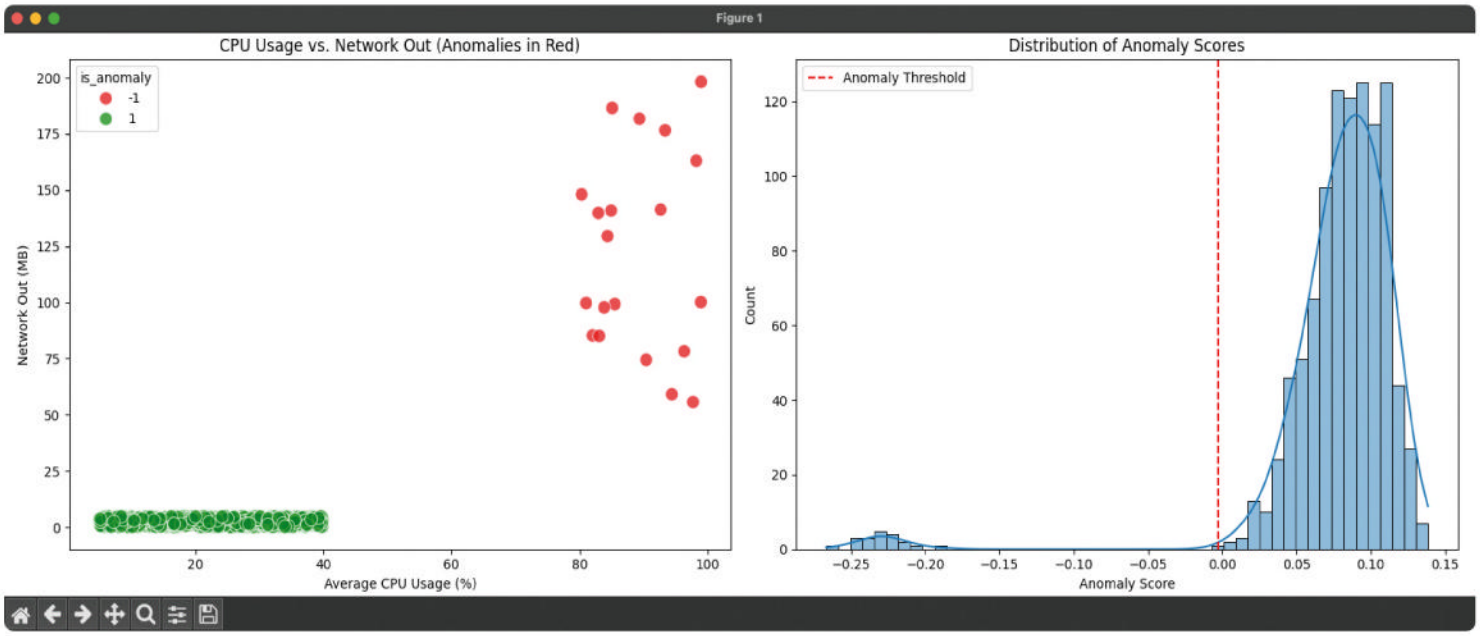


Figure 3: Visualization of the anomalies

You could add alerting mechanisms. In the main function, I've left a commented-out section for send_alert. In a production system, this could integrate with a messaging service such as Slack or PagerDuty, a SIEM (Security Information and Event Management) system, or simply log to a persistent file.

For a real-world system, you'd probably want a continuous monitoring system, so you wouldn't have to retrain the model every time new data shows up. For that, you'd want to look into model persistence. You'd train it once, save it (joblib.dump(model, MODEL_PATH)), and then load it for predictions (joblib.load(MODEL_PATH)).

And of course, you could look into leveraging more sophisticated models. For more complex data, you might explore other unsupervised anomaly detection algorithms like Local Outlier Factor (LOF), One-Class SVM, or even deep learning approaches like autoencoders.

Summary

It's simply amazing what we can achieve as developers today. The journey from a simple tweet to a thought to a completely functioning code example just took me a few hours. Converting it into a real-world application that can actively start detecting threads isn't that much of a leap. I've built a fully functional, fundamental anomaly detection system, running entirely on my off-the-shelf commercial macOS machine!

This project demonstrates how machine learning, even in its simplest forms, can be a powerful ally in the fight against cyber threats. By understanding "normal" and flagging "abnormal," you're equipped to detect new and evolving dangers that traditional methods might miss. This local set up provides a fantastic sandbox for further experimentation, allowing you to explore the fascinating intersection of ML and security right from your developer workstation.

Listing 7: Putting together the anomaly detection logic

```
def main():
    # 1. Load Data
    try:
        df = pd.read_csv(DATA_FILE)
        print(f"Loaded data from {DATA_FILE}.
          Shape: {df.shape}")
    except FileNotFoundError:
        print(f"Error: {DATA_FILE} not found.")
        return

    # 2. Train Model
    model = train_model(df)

    # 3. Predict Anomalies
    # Pass the original DataFrame here
    df_results = predict_anomalies(model, df)

    # 4. Display Anomalies
    anomalies = df_results[df_results['is_anomaly'] == -1]
    normal_data = df_results[df_results['is_anomaly'] == 1]

    print(f"\nFound {len(anomalies)} anomalies
         and {len(normal_data)} normal data points.")
    if not anomalies.empty:
        print("\n--- Detected Anomalies ---")
        # Show top 5 most anomalous
        print(
         anomalies.sort_values(by='anomaly_score').head())

        # here you can implement an alerting mechanism
        # for _, row in anomalies.iterrows():
        #     send_alert(..)
    else:
        print("\nNo anomalies detected.")

    # 5. Visualize Results
    visualize_results(df_results)
```

I'm curious. How are you using AI in your real-world applications? From what I can tell, the true potential of AI is only beginning to be tapped. What do you think?

Until next time, happy coding. Securely of course.

Sahil Malik

Angular Signals in the Real World: Smarter Inputs and Reactive Routing

In Part 1 of this series, I explored the foundations of Angular Signals—the reactive primitives signal(), computed(), and effect()—and how they simplify local state management and change detection. Those concepts form the core of a signal-driven architecture, but production applications require more than just reactive counters or basic state. In this follow-up, you'll step into real-world scenarios where Signals fundamentally change how you design Angular applications. You'll see how to replace lifecycle-bound input handling with live, reactive @Input() values, remove boilerplate from @ViewChild and @ContentChild queries, modernize two-way binding with model(), build guards that use signals and route parameters to make navigation decisions at the moment a route is activated, and connect those same parameters to httpResource() for declarative, signal-powered API calls.

Along the way, you'll connect these concepts so that data can flow seamlessly from route parameters to component inputs and straight into live API resources—keeping your UI and back-end in sync with minimal glue code.

Sonu Kapoor
Sonukapoor@gmail.com
linkedin.com/in/sonu-kapoor
@sonukapoor1978

Sonu Kapoor has been a software developer since 2003. He's currently focused on web development using the Angular Framework and is a Microsoft MVP and a GDE in Angular. Sonu has written numerous articles that are available on the internet.

Sonu is the author of "Beginning JavaScript Syntax: Understanding Syntactical Rules and Structures for Better JavaScript Programming," published by Apress in 2025.

Using Signals as Component Inputs

One of the most impactful additions in Angular's Signals ecosystem is the ability to pass Signal<T> values into components via the input<T>(). This isn't just a new syntax; it changes the way you structure component communication entirely.

Traditionally, Angular developers have had to rely on lifecycle hooks like ngOnChanges, custom setters, or even RxJS subjects to react to input changes. These patterns worked, but they were verbose, fragile, and often error-prone, especially as the number of inputs grew.

The Pain of ngOnChanges: A Familiar Mess

Let's say you're building a configuration panel for an enterprise dashboard. You receive several inputs from the parent component, and these influence what sections are displayed, what themes apply, and what permissions unlock advanced features.

Listing 1 shows how that looked in pre-Signals Angular using ngOnChanges:

Yes, it works. But it's fragile:

- You're forced to repeat conditional logic based on SimpleChanges.
- Derived state logic like showLegalBanner becomes entangled with imperative state tracking.
- Any missed dependency or unguarded usage can cause subtle bugs.
- You need to manually ensure that values are defined (featureFlags might be undefined the first time).
- Every new @Input() increases complexity non-linearly.

This sort of pattern is common in large applications, and it becomes a nightmare to test, debug, and refactor.

Signals: Declarative by Design

Listing 2 shows the same use case, but with Signals:

Now each piece of state is expressed declaratively as a function of its inputs. Angular ensures that when any input signal changes, the relevant computed values are updated automatically, no boilerplate required.

This isn't just a cleaner syntax. It's a new contract between components:

- Inputs are live values.
- Derived state is explicit.
- Updates are automatic and scoped to what changes.

Queries and Content Children Are Also Reactive

In traditional Angular, querying for child components, elements, or projected content meant reaching for decorators like @ViewChild, @ContentChild, and their plural forms. These queries weren't reactive; they were static lookups tied to Angular's component lifecycle.

To safely access them, developers had to wait for ngAfterViewInit() or ngAfterContentInit(), and even then, conditional rendering (*ngIf/@if) could cause queries to resolve to be undefined unexpectedly.

Before: Lifecycle-Dependent @ViewChild

```
@ViewChild(CustomCardHeader)
header!: CustomCardHeader;

ngAfterViewInit() {
  if (this.header) {
    this.header.setFocus();
  }
}
```

This pattern forced developers to:

- Wait for lifecycle hooks to fire.
- Handle nullish values manually.
- Often write setTimeout or use ChangeDetectorRef to avoid ExpressionChangedAfterItHasBeenChecked errors.

Signal-Based viewChild()

With Angular Signals, you can now use the viewChild() query function, which returns a **Signal<T | undefined>**, making queries truly reactive.

Listing 1: Pre-Signals using ngOnChanges

```
@Component({
  selector: 'legacy-config-panel',
  template: `...`
})
export class LegacyConfigPanel
  implements OnChanges {
  @Input() showAdvanced!: boolean;
  @Input() theme!: 'light' | 'dark';
  @Input() userRole!: string;
  @Input() region!: string;
  @Input() featureFlags!: string[];

  isAdminView = false;
  isDarkMode = false;
  showExperimental = false;
  regionSupportsFeatureX = false;
  showLegalBanner = false;

  ngOnChanges(changes: SimpleChanges): void {
    if (changes['userRole']) {
      this.isAdminView = this.userRole === 'admin';
    }

    if (changes['theme']) {
      this.isDarkMode = this.theme === 'dark';
    }

    if (changes['userRole'] ||
        changes['showAdvanced']) {
      this.showExperimental =
        this.userRole === 'admin' &&
        this.showAdvanced;
    }

    if (changes['region'] ||
        changes['featureFlags']) {
      this.regionSupportsFeatureX =
        this.region === 'EU' && this.featureFlags?.includes('feature-x');
    }

    if (
      changes['region'] ||
      changes['userRole'] ||
      changes['featureFlags']
    ) {
      this.showLegalBanner =
        this.region === 'EU' &&
        this.userRole !== 'admin' &&
        !this.featureFlags?.includes('suppress-legal');
    }
  }
}
```

Listing 2: Using Signals instead of ngOnChanges

```
@Component({
  selector: 'config-panel',
  template: `...`
})
export class ConfigPanel {
  showAdvanced = input<boolean>();
  theme = input<'light' | 'dark'>();
  userRole = input<string>();
  region = input<string>();
  featureFlags = input<string>();

  readonly isAdminView = computed(() =>
    this.userRole() === 'admin');

  readonly isDarkMode = computed(() =>
    this.theme() === 'dark');

  readonly showExperimental = computed(() =>
    this.userRole() === 'admin' &&
    this.showAdvanced()
  );


  readonly regionSupportsFeatureX = computed(() =>
    this.region() === 'EU' &&
    this.featureFlags()?
      .includes('feature-x')
  );

  readonly showLegalBanner = computed(() =>
      this.region() === 'EU' &&
      this.userRole() !== 'admin' &&
      !this.featureFlags()?
        .includes('suppress-legal')
  );
}
```

```
header = viewChild(CustomCardHeader);
readonly hasHeader = computed(() =>
                      !!this.header());
```

This eliminates the need for lifecycle timing. You can use the query directly inside computed() or effect(), and Angular tracks it reactively, updating as soon as the queried component enters or leaves the DOM.

You no longer have to remember whether @ViewChild is available in ngAfterViewInit vs. ngOnInit. Instead, you think in terms of reactivity, and your component becomes simpler, more testable, and less error-prone.

*A Note on *ngIf/@if and Conditional Presence*

One important caveat: Queries only resolve when their targets exist in the DOM. If the queried element or component is inside an *ngIf/@if, its signal will return undefined until the condition is true.

This behavior is identical to the old @ViewChild approach but because signals are reactive, you no longer need to manage lifecycle edge cases manually. You can write logic like:

```
effect(() => {
  if (this.header()) {
    this.header()!.setFocus();
  }
});
```

This code will re-run whenever the header becomes available. No lifecycle hook or ngIf timing logic required.

Query What You Want, Reactively

Signal-based queries:

- Eliminate lifecycle hooks (ngAfterViewInit, etc.)
- React to structural DOM changes like *ngIf/@if and *ngFor/@for

- Improve ergonomics and safety in dynamic layouts
- Work seamlessly inside computed() and effect()

Just like with signal-based inputs, the move from @ViewChild to viewChild() shifts component design from imperative and lifecycle-bound to reactive and declarative, in line with modern Angular architecture.

Signal-Based Two-Way Binding: model() Inputs

Two-way binding in Angular has historically required a manual pairing of @Input() and @Output() properties or, for more complex use cases, the implementation of ControlValueAccessor to integrate with Angular's forms API.

Although both approaches work, they introduce ceremony, tight coupling, and an imperative style that doesn't align well with Angular's new signal-first reactivity model.

Angular's new model() API offers a simpler, reactive alternative. It enables components to support [(value)] syntax by exposing a single WritableSignal property with no need for paired decorators or form adapters.

Let's take a look at how this looks using the standard input/output pattern.

```
@Component({ /* ... */ })
export class LegacySlider {
  @Input() value = 0;
  @Output() valueChange =
    new EventEmitter<number>();

  increment() {
    this.valueChange.emit(this.value + 10);
  }
}
```

In the parent:

```
<legacy-slider [(value)]="volume" />
```

And the following is required:

- Coordinating names like value and valueChange
- Manually emitting changes
- Repeating the same boilerplate across components

Using Signal-Powered model() Input this makes the implementation much simpler.

```
@Component({ /* ... */ })
export class CustomSlider {
  value = model(0); // Declare a signal-based model
     input
  increment() {
    this.value.update(old => old + 10);
  }
}
```

In the parent:

```
@Component({
  template: `<custom-slider [(value)]="volume" />`
})
export class MediaControls {
  volume = signal(0);
}
```

The model() API handles:

- Input binding ([value])
- Output emission ((valueChange))
- Writable signal access (value() and value.set(...))

This approach is clean, reactive, and fully compatible with Angular's signals ecosystem.

A Note on ControlValueAccessor

To be clear: The model() API does not replace ControlValueAccessor—at least not yet.

CVA is still required when integrating a custom component with Angular's FormControl, FormGroup, or formControlName. However, in many real-world cases, developers reached for CVA just to enable two-way binding or to make a reusable toggle, slider, or checkbox "form compatible."

With model(), many of these use cases no longer need CVA at all.

If you're building standalone form components, internal UI elements, or a design system not tightly coupled to Angular Forms, the model() API offers a much simpler and more declarative way to bind state between parent and child, without lifecycle hooks, without extra output events, and without implementing an interface.

Route Params in Guards and Components (the Signal-Friendly Way)

Angular's router doesn't expose route params as signals inside guards yet. Guards run per navigation and must return a plain value (Boolean | UrlTree | Observable | Promise). You can still use your app's signals (e.g., auth state) inside guards, but read params from the guard arguments (snapshots). For components, you can receive params as signals via withComponentInputBinding().

Using Route Params in Guards (correct)

```
import { CanActivateFn } from '@angular/router';
import { inject } from '@angular/core';
import { AuthService } from './auth.service';

export const projectAccessGuard:
  CanActivateFn = (route) => {
  // may expose signals like isAdmin()
  const auth = inject(AuthService);

  // string | null from snapshot
  const projectId = route
    .paramMap
    .get('projectId');

  // return boolean/UrlTree/Observable/Promise
  if (!projectId) return false;
  return auth.hasAccessToProject(projectId);
};
```

It's important to be clear about what Signals can and can't do inside guards. Guards always run once per navigation and must return a concrete result–not a Signal.

Listing 3: Using withComponentInputBinding

```
@Component({
  standalone: true,
  selector: 'project-details',
  template: `
    @if (loading()) {
      <p>Loading…</p>
    }

    @if (error()){
        <p>Error loading details.</p>
        <p>{{ error() }}</p>
      }

    @if (data(); as project) {
        <h1>{{ project.name }}</h1>
        <p>{{ project.description }}</p>
      }
    }
  `
})
export class ProjectDetailsComponent {
  projectId = input.required<string>();

  resource = httpResource(() =>
    return `/api/projects/${this.projectId()}`
  );

  data = this.resource.data;
  loading = this.resource.loading;
  error = this.resource.error;
}
```

- Guards don't return signals.
- If your auth/permission model uses signals, read their current value inside the guard and return a concrete result.
- Guards don't auto re-run when signals change; they evaluate on each navigation.

Passing Route Params to Components as Signals

For routed components, enable withComponentInputBinding() so params, query params, and route data map directly to inputs as signals-no ActivatedRoute, no subscriptions.

In your app.config.ts:

```
import { ApplicationConfig } from '@angular/core';
import {
  provideRouter,
  withComponentInputBinding
} from '@angular/router';
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes,
      withComponentInputBinding()),
  ],
};
```

In the component:

```
import { Component, input } from '@angular/core';

@Component({
  standalone: true,
  selector: 'project-details',
  template: `Project ID: {{ projectId() }}`,
})
export class ProjectDetailsComponent {
  // Matches ':projectId' by name; provided as a
  // Signal<string>
  projectId = input<string>();
}
```

That's it. Angular will:

- Match :projectId to projectId
- Provide it as a signal (projectId())
- Keep it updated if the route changes

You can add more inputs to bind query params and route data simply by name (e.g., ref = input<string | undefined>();, mode = input<'readonly' | 'edit'>();).

Loading API Data with httpResource() and Signals

One of the most compelling use cases for Signals is wiring them directly into API calls, and Angular's experimental httpResource() API makes this frictionless. Instead of juggling loading flags, subscription lifecycles, and error handling yourself, httpResource() produces three ready-to-use signals: data, error, and loading.

Let's extend the earlier example where the projectId comes from withComponentInputBinding() (**Listing 3**). Because projectId is already a signal, you can feed it straight into httpResource() to create a live, reactive API resource:

Here, both the route param (projectId) and the API resource are signals. When projectId() changes (for example, when navigating between projects), Angular automatically re-runs the resource factory, triggering a new HTTP request without any imperative glue code.

This pattern scales naturally: You can add derived state with computed() over data(), implement stale-while-revalidate caching, or hydrate the resource server-side with Angular Universal's TransferState for instant render. In all cases, the flow remains declarative: You describe what data you want and Angular decides when to fetch it.

Because httpResource() produces data, loading, and error as signals, you can compose them with other signals in your application without any additional wiring. This makes it easy to derive view models—such as filtered lists, computed summaries, or "is stale" flags—directly from the resource state.

In production, this approach opens the door to richer patterns:

- **Parameter-driven resources:** Passing route parameters or other signals into the resource factory lets you automatically re-fetch when those inputs change. In this example, switching the projectId param immediately triggers a fresh HTTP request with no manual refresh logic.

- **Layered UX states:** You can show cached or last-known data while a background request runs, display skeletons for new views, or swap to error messaging without losing context.
- **SSR and hydration:** With Angular Universal, you can resolve the resource on the server, embed the state into the HTML, and have the client hydrate it without refetching. This reduces time-to-first-render and prevents duplicate HTTP calls.
- **Error handling and recovery:** Because error() is a signal, you can reactively surface retry buttons, log issues, or adjust the UI flow based on error type, all without special-case state flags.

The biggest advantage is that your API state becomes a first-class part of your reactive graph. Instead of manually synchronizing API results with local variables, httpResource() keeps the source of truth in one place, allowing the rest of your application to react naturally as that state changes. This is where Signals truly shine, not just in holding values, but in orchestrating how those values flow through the UI.

Choosing the Right Context for httpResource()

Although httpResource() is a powerful way to manage API state as part of your reactive graph, it's not a one-size-fits-all solution. In practice, you'll want to be intentional about where and how it's introduced.

It shines in UI-driven, read-heavy scenarios, for example, loading detail views from route parameters, populating drop-downs, or powering dashboards that need clear loading and error states. The fact that it produces structured signals makes it ideal for components or services that feed directly into templates.

However, some workflows may call for different patterns:

- **Long-lived, shared state:** If multiple, unrelated features need the same API data, you may prefer a dedicated service with a single resource instance instead of creating a new one per component.
- **Streaming or push data:** For websockets, server-sent events, or other continuous data streams, RxJS operators may still be the better fit, with a conversion to signals for UI consumption.
- **Complex mutation workflows:** For multi-step forms, optimistic updates, or transactions that depend on several writes, httpResource() might be just one piece of a larger service orchestration.

The key is to treat httpResource() as a building block in your architectural toolbox. Use it where its built-in state tracking reduces boilerplate and improves clarity, and don't hesitate to combine it with other reactive primitives or patterns when your requirements extend beyond a simple "fetch-show-refresh" cycle.

Summary

Signals are more than a new syntax for state: They enable a different way of thinking about Angular architecture. By using them for component inputs, queries, two-way bindings, routing context, and API calls, you can replace imperative state tracking and lifecycle hooks with declarative, reactive expressions that update automatically and only when needed.

The examples in this article show how to:

- Treat @Input() values as live signals instead of snapshot props
- Query child components reactively with viewChild() and contentChild()
- Simplify two-way binding with model() without boilerplate or CVA (in many cases)
- Build guards that use signals and route parameters to decide on navigation at activation time
- Pass route parameters directly to components as signals
- Feed route parameters into httpResource() to build live, reactive API resources

Together, these patterns let you move from snapshots and subscriptions to declarative state graphs, making your Angular apps cleaner, more predictable, and easier to maintain at scale.

Sonu Kapoor

SPONSORED SIDEBAR:

Stay Connected with CODE Magazine on LinkedIn

Join our professional community on LinkedIn to stay ahead in the tech world.

Why Connect with Us?

Exclusive Content: Get the latest tech insights and articles.

Industry Updates: Stay informed with cutting-edge news.

Networking: Connect with peers and industry leaders.

Help us grow by sharing with colleagues and friends.

Follow us today!
www.linkedin.com/company/**code-magazine**

CODE CONSULTING

APPS WITHOUT COPILOTS ARE NOW LEGACY!

Microsoft has introduced Copilot-driven applications and announced that all Microsoft applications will follow this approach going forward. This aims to capitalize on the extreme productivity gains this new paradigm promises. The same applies to your own applications and systems.

We can help you build Copilots into your own applications.atered to what you do.

VISIT OUR COPILOT PAGE TO FIND OUT MORE ABOUT OUR SERVICES AND TO SEE DEMONSTRATIONS AND EXAMPLES!

codemag.com/copilot

832-717-4445 ext. 9 • info@codemag.com

©shutterstock

Integrating AI and ML in .NET Core with ML.NET

In this ever-changing era of rapid technological innovations and advancements, machine learning (ML) and artificial intelligence (AI) are revolutionary technologies that have been changing industries by enabling data driven automation, better user experiences, and predictive analytics. ML.NET is Microsoft's cross-platform, open-source machine learning framework that

brings AI and ML to.NET developers. ML.NET lets you build, train, and deploy machine learning models that can help you build high-quality intelligent applications that can cater to the needs of today's businesses. It enables you to solve complex problems like classification, regression, clustering and recommendation using tools you've already used or that you're aware of as a .NET developer.

You can use any of the programming languages supported by .NET and the programming editor you're familiar with (i.e., Visual Studio) to build intelligent applications by leveraging machine learning algorithms from within the .NET ecosystem. For example, you can write your programs in C# or F# from within your Visual Studio IDE that can take advantage of machine learning algorithms to solve complex problems.

Some of the most popular ML frameworks include:

- TensorFlow
- PyTorch
- Microsoft CNTK
- Amazon Machine Learning

This article presents a discussion of AI and ML and how they can be integrated in .NET Core applications. To work with the code examples discussed in this article, the following software must be available:

- Visual Studio 2022 Preview
- .NET 10.0 Preview
- ASP.NET 10.0 Preview Runtime

> At the end of this journey, you'll be able to build intelligent applications using ML.NET in ASP.NET Core.

You can use ML.NET to do most of the things you can do with Azure Cognitive Services. ML.NET lets you work with machine learning in C# or F# without leaving the .NET ecosystem. You can use ML.NET to integrate machine learning into your web, mobile, desktop, and even IoT apps. Here's what you can do with ML.NET at a glance:

- **Regression:** Helps predict numerical values (i.e., item prices, population, etc.)
- **Classification:** Sorts data into categories (i.e., spam detection, fraud detection)
- **Clustering:** Groups similar data points (i.e., customer segmentation)
- **Recommendation:** Provides suggestions on products or content
- **Anomaly detection:** Identifies unusual patterns in data (i.e., fraud detection)
- **Time series forecasting:** Predicts future values based on a set of historical data (i.e., stock market predictions)
- **Model evaluation:** Assesses model performance based on their accuracy, precision, etc.

The key features of ML.NET include:

- It's a cross-platform framework, enabling you to use it in Windows, macOS, and Linux operating systems.
- It's open-source and community-driven.
- It provides excellent support for extensibility.
- It's high-performant and scalable.
- It includes support for integration with existing ML frameworks.
- It has seamless integration with the .NET ecosystem.

The key reasons to integrate ML.NET into your ASP.NET Core applications:

- Enhanced efficiency
- Improved accuracy
- Process automation
- Predictive analytics
- Advanced analytics
- Better analytical predictions
- Enhanced customer experience
- Enhanced decision-making

A Typical Machine Learning Workflow

A typical machine learning workflow encompasses the following phases:

1. **Problem statement:** Define the business problem and then analyze how best to apply machine learning to solve the problem.
2. **Data collection:** The first phase in this workflow retrieves data from various data sources.
3. **Data preparation:** The data is cleaned to remove any errors and make the data ready for analysis and modeling.
4. **Model selection and training:** Select the appropriate machine learning algorithm (i.e., decision tree, logistic regression, neural network, etc.) and train the model using the dataset so that it knows the patterns and relationships in the data to be able to make predictions.
5. **Evaluation:** Test the trained model to measure performance using metrics such as precision, accuracy, etc.
6. **Deployment:** The trained and validated model is integrated into the production systems (i.e., business applications built using ASP.NET Core, mobile apps, desktop apps, etc.) so that it can generate predictions or automate decisions in real world environments.
7. **Monitoring and maintenance:** Monitor the model performance, and retrain and update the model as needed to keep it accurate and relevant over time.

Understanding the ML.NET Components

The core components of ML.NET include:

- ML Context
- IDataView
- Data loaders and data converters
- Model training
- Model evaluation

Joydip Kanjilal
joydipkanjilal@yahoo.com

Joydip Kanjilal is an MVP (2007-2012), software architect, author, and speaker with more than 20 years of experience. He has more than 16 years of experience in Microsoft .NET and its related technologies. Joydip has authored eight books, more than 500 articles, and has reviewed more than a dozen books.

The **MLContext** represents the starting point in any ML.NET workflow and provides a single context to manage data loading, processing, transformation, model training, evaluation, and model persistence.

In ML.NET, data is represented using an abstraction named **IDataView**. It provides an efficient and schema-aware representation of both training and prediction data. It should be noted that IDataView objects are evaluated lazily for better performance and resource efficiency.

When working with ML.NET, you often need to retrieve data from similar or disparate data sources. For example, these data sources can contain data in identical data formats or different data formats. ML.NET offers excellent support for **loading data** pertaining to varied file formats, including CSV and text files. You can load data into an IDataView from any data source.

You can either use File Loaders for retrieving data available in text, binary, and image files, or, use the Database Loader to load data from a database and then train the datasets. Data Transformers are used to clean and transform the data in a format that's understandable by ML.NET.

Model training is a process used to build and train the models. ML.NET takes advantage of one or more of the built-in algorithms for training models.

- Clustering
- Classification
- Regression
- Time series analysis
- Anomaly detection

Before you deploy a trained model into the production environment, you want to make sure that it's high-quality so that it can make the right predictions using **model evaluation**. ML.NET has multiple evaluators to evaluate the accuracy, precision, and performance of trained models. These model evaluators take advantage of several metrics to measure how well your model is performing in real-time. The most commonly used metrics in this regard are:

- **Binary classification:** Accuracy, precision, recall, F1Score, AreaUnderROC
- **Regression:** RMSE (root mean squared error), RSquared
- **Clustering:** Normalized mutual information

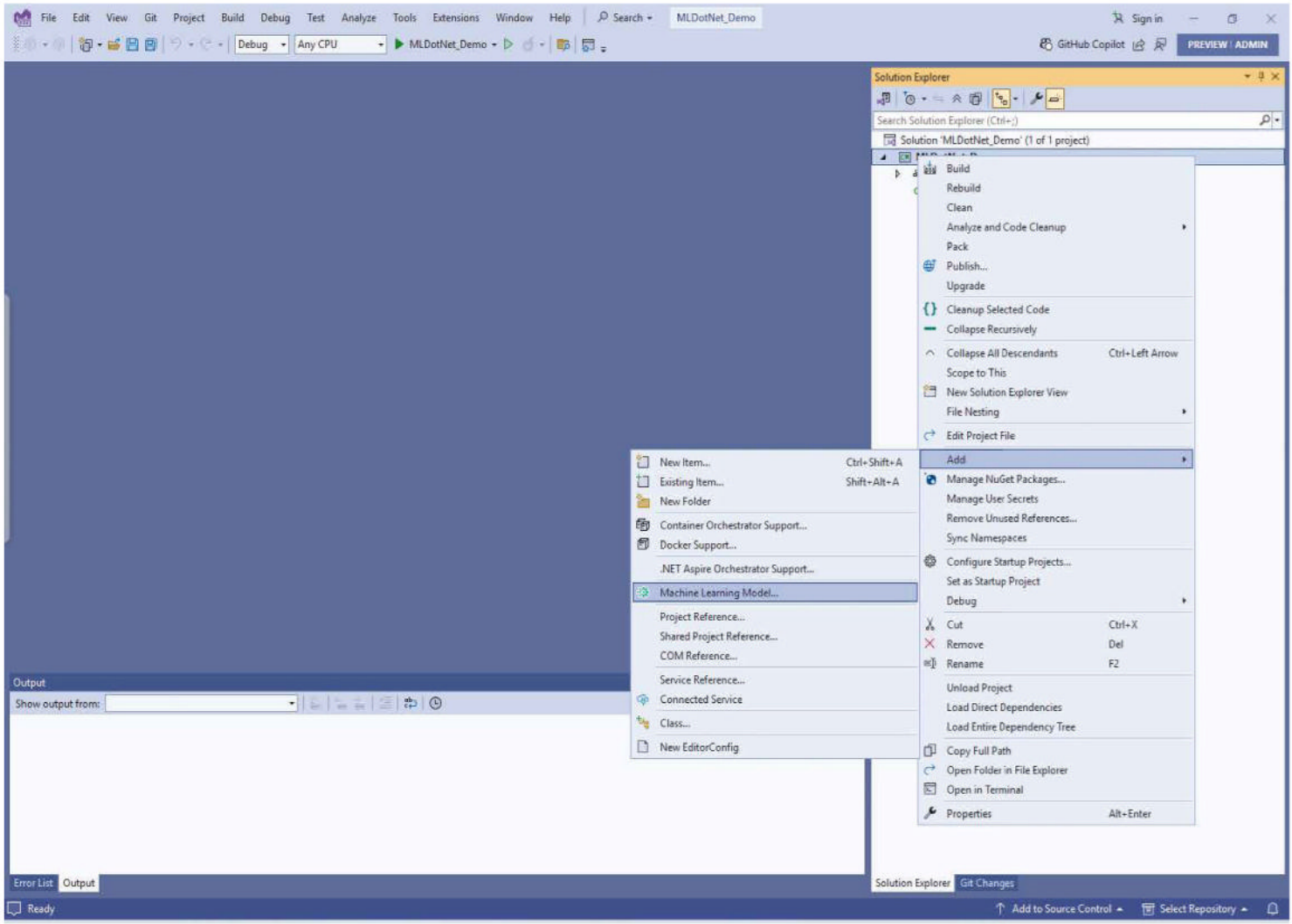


Figure 1: Adding machine learning support to the project

Getting Started with ML.NET in Visual Studio

In this section, I'll examine how you can create a simple application using ML.NET. I'll start by creating a console application in Visual Studio 2022.

Create a New .NET 10 Console Application

Let's create a Windows application project to work with the C# 14 code examples given in the subsequent sections of this article. You can create a WPF project in Visual Studio 2022 in several ways. When you launch Visual Studio 2022, you'll see the Start window. You can choose **Continue without code** to launch the main screen of the Visual Studio 2022 IDE.

To create a new Windows Application Project in Visual Studio 2022:

1. Start the Visual Studio 2022 Preview IDE.
2. In the Create a new project window, select **Console App** and click Next to move on.
3. Specify the project name and the path where it should be created in the **Configure your new project** window.
4. If you want the solution file and project to be created in the same directory, optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
5. In the Additional information screen, specify the target framework as .NET 10.0 (Preview).
6. Click Create to complete the process.

Add Machine Learning Capabilities to Your Project

Next, right-click on the project in the Solution Explorer window and then select **Add > Machine Learning Model** as shown in **Figure 1**.

In the Add New Item window that's shown next, ensure that you've selected the Machine Learning Model (ML.NET) template. Then, specify a name for the model config file and click on the Add button. Note that the extension of this file should be mbconfig. The mbconfig file, as the name suggests, is a configuration file based on JSON format that contains the state and configuration information of your machine learning model project.

In the Add New Item window that's shown next, ensure that you've selected the Machine Learning Model (ML.NET) template. Then, specify a name for the ML model config file and click on the Add button, as shown in **Figure 2**.

Let's now implement a typical ML workflow in a step-by-step manner.

Creating a Model Using the ML.NET Model Builder

The ML.NET Model Builder is a graphical extension that you can use inside your Visual Studio IDE, and which al-

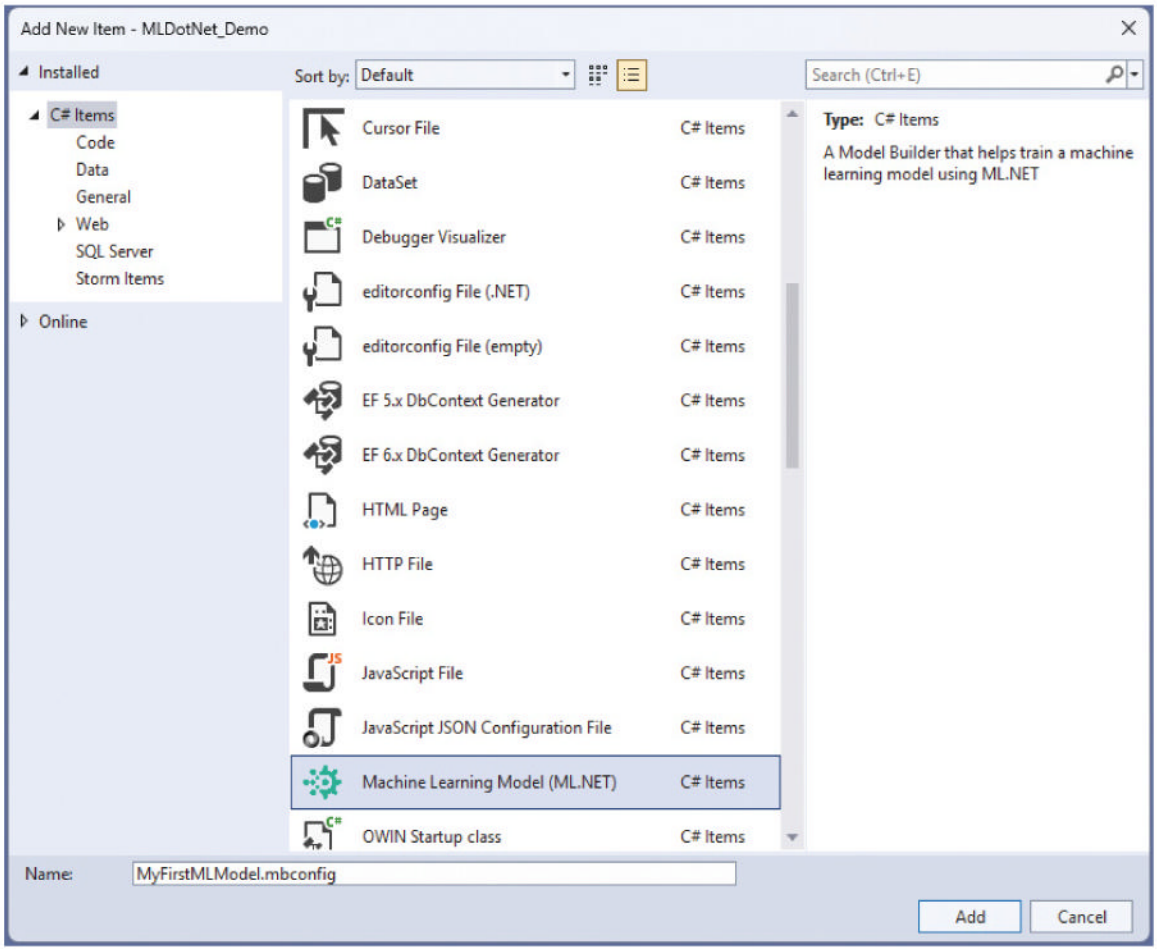


Figure 2: Adding the ML Model to the project

lows .NET developers to build, train, and deploy custom machine learning models without needing to have any prior knowledge of machine learning.

Here are some of its key features:

- **Graphical interface:** The ML.NET Model Builder provides a wizard-based GUI tool, so you don't need to write code or know ML to start building models—just provide your data and choose your scenario.
- **Automated machine learning (AutoML):** The Model Builder uses AutoML to try different ML algorithms, settings, and parameters to find the best model for you.
- **Scenario selection:** The Model Builder provides support for common ML tasks like classification, value (regression) prediction, recommendation, forecasting, image classification, and more. You choose what you want to predict, and Model Builder takes care of the rest.
- **Data connection:** The Model Builder can load data from CSV, TXT, and SQL Server databases.

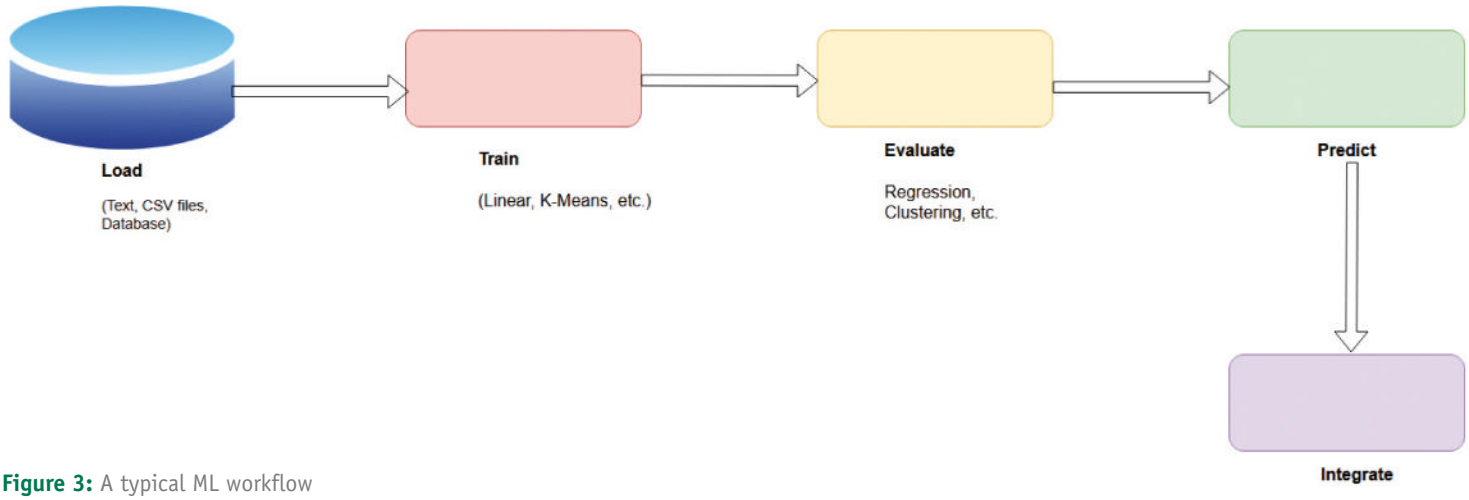


Figure 3: A typical ML workflow

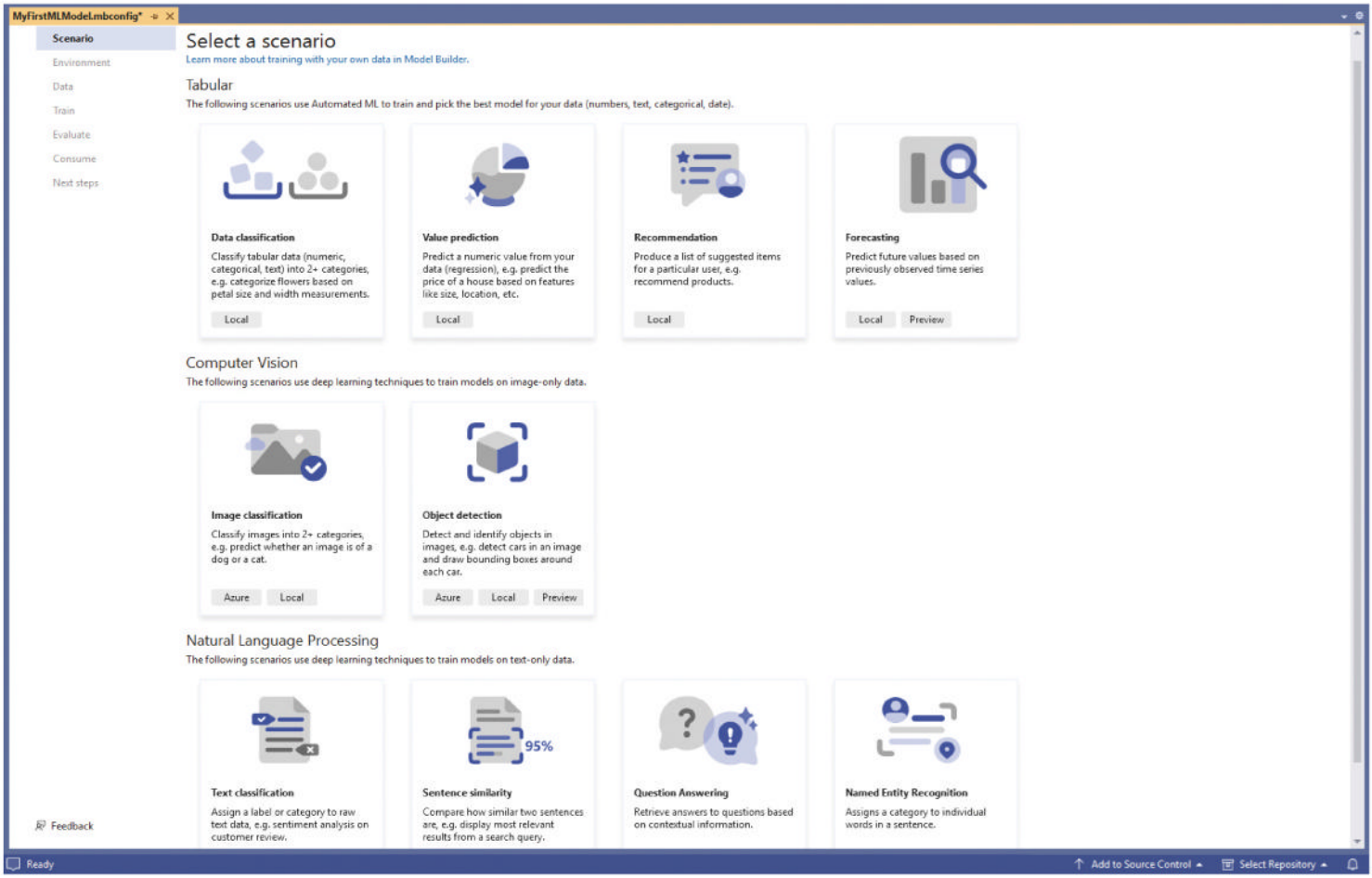


Figure 4: Selecting a machine learning scenario

- **Code generation:** After successful model training, the Model Builder generates code files for model training and consumption.
- **Integration and deployment:** You can add the generated code and models to any of your .NET projects.
- **Free:** You don't incur any additional costs to use the Model Builder tool because it's freely available by default in your Visual Studio IDE.

When you're using the Model Builder, follow these steps to build and train your ML models:

- Choose an ML scenario.
- Select your data.
- Train the model,
- Evaluate.
- Predict.
- Generate code.

Figure 3 illustrates the complete ML workflow.

Choose a Machine Learning Scenario

To build a model using the model builder, start by selecting a machine learning scenario. In this example, select the Data classification scenario from the Model Builder window from the Select a scenario screen, as illustrated in **Figure 4**.

Select the Training Environment

Once you've selected the scenario, choose the training environment. Select **Local (CPU)** as the training environment for your model, as shown in **Figure 5**.

Adding Data

To add data for your model, select the data source type for your input data (i.e., the CSV file in the local file system in this example) and then load the CSV file. Additionally, also select the column you want to predict, i.e., select the desired column to use for prediction by specifying the Column to predict (Label). In this example, I've selected the Sentiment column, as shown in **Figure 6**.

You can use the advanced options window to modify column settings, schema information, and configure test data.

Train the Model

In the next step, you need to train the model with the data you loaded from the CSV file. The Model Builder is adept at training the model automatically but you can also customize it by specifying the training duration and a few other additional parameters via the Advanced training options screen. Once you've specified the custom training duration and then clicked on the Train again button, the training results will be displayed, as shown in **Figure 7**.

Evaluate the Model

Once you've trained the model successfully, you can evaluate the model to make predictions on a piece of test data and then access the results. The Model Builder provides an intuitive interface to enter the input data and then see how the model can predict data, as shown in **Figure 8**.

Consume the Model

Finally, it's time to consume the model. The Model Builder helps you generate two types of applications to consume the model: either generate a Console application or create a Web Api application. Refer to **Figure 9**, which illustrates how the Model Builder uses in-built project templates to generate these applications for you.

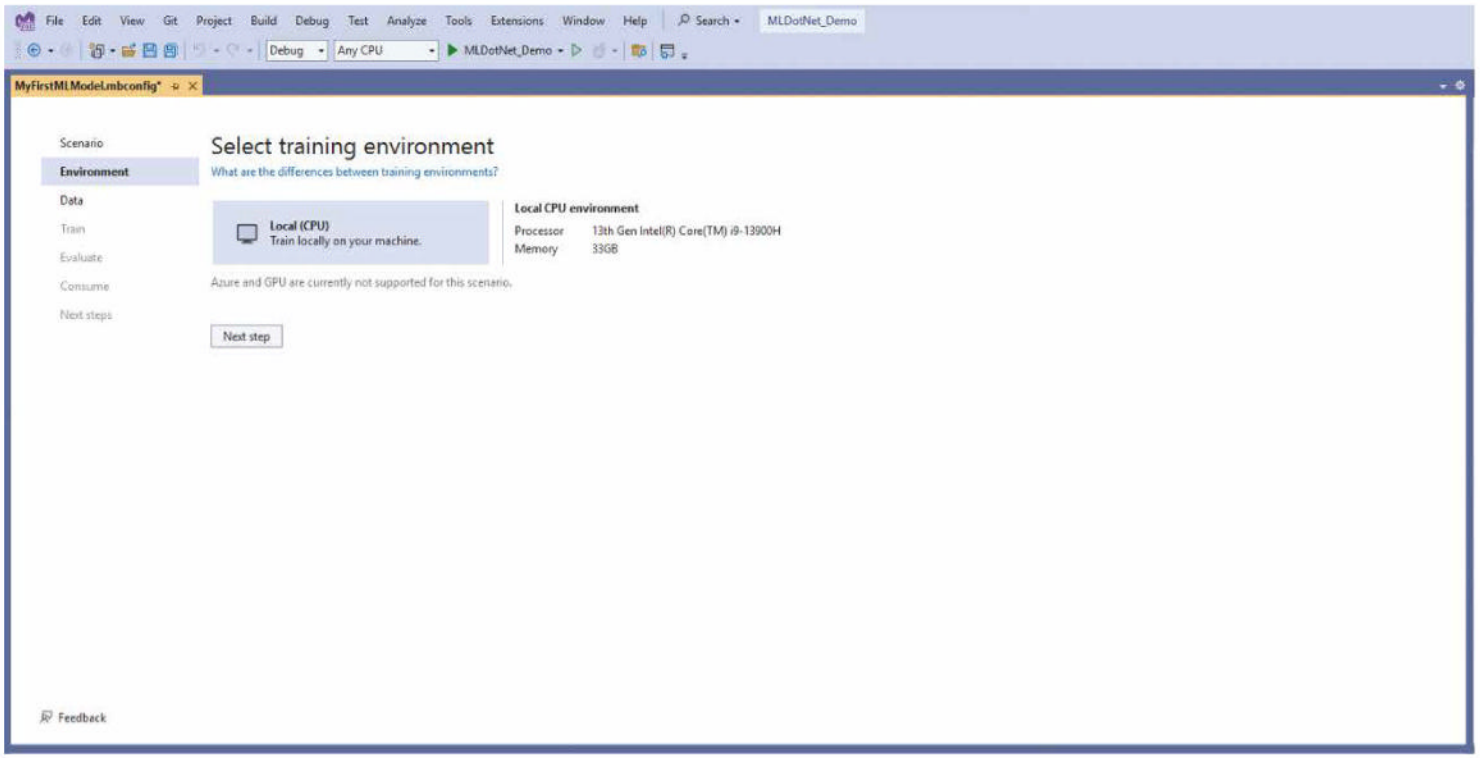


Figure 5: Select a training environment for your model.

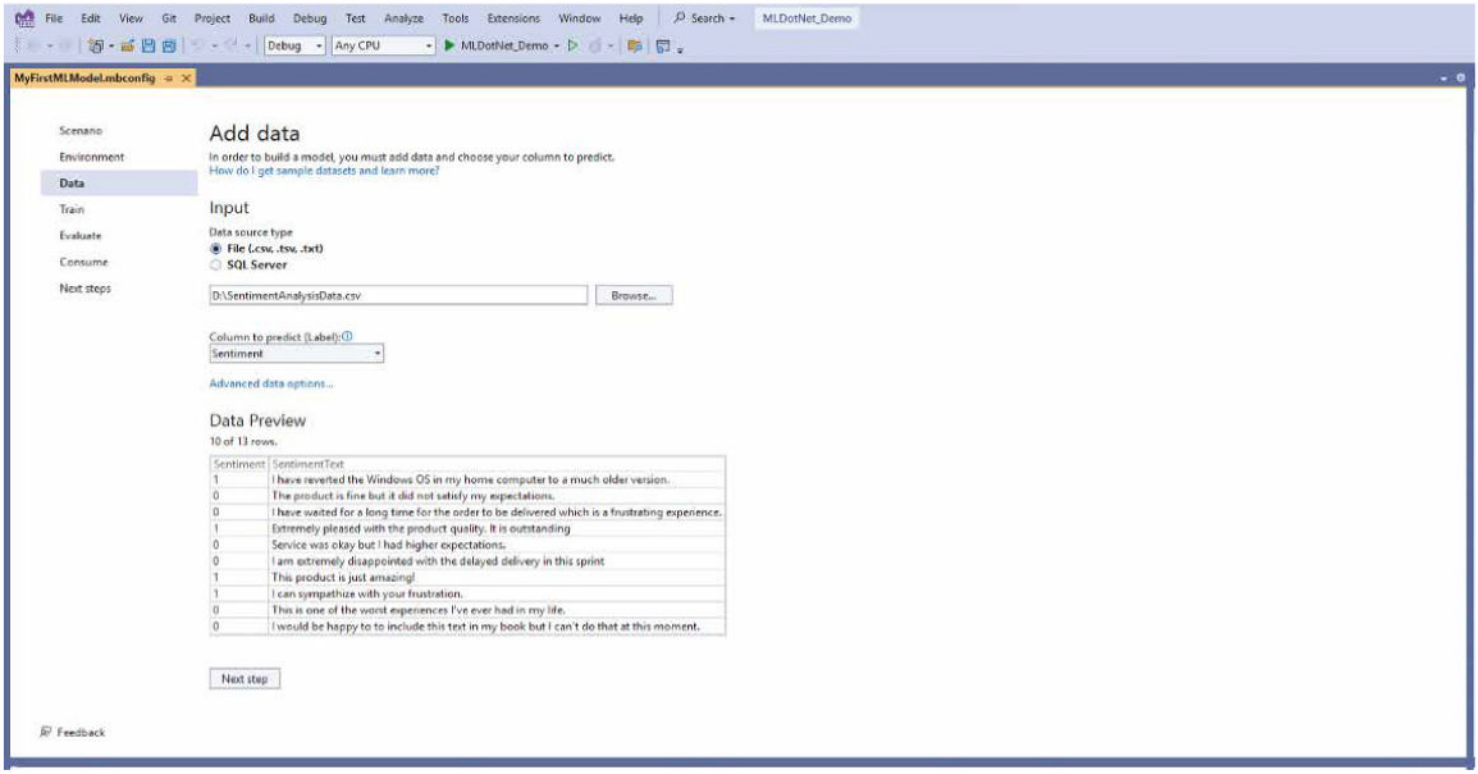

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
MLDotNet_Demo
Debug
Any CPU
MLDotNet_Demo
MyFirstMLModel.mbconfig
Scenario
Environment
Data
Train
Evaluate
Consume
Next steps
Add data
In order to build a model, you must add data and choose your column to predict.
How do I get sample datasets and learn more?
Input
Data source type
File (.csv, .tsv, .txt)
SQL Server
D:\SentimentAnalysisData.csv
Browse...
Column to predict (Label):
Sentiment
Advanced data options...
Data Preview
10 of 13 rows.
Sentiment
SentimentText
1 I have reverted the Windows OS in my home computer to a much older version.
0 The product is fine but it did not satisfy my expectations.
0 I have waited for a long time for the order to be delivered which is a frustrating experience.
1 Extremely pleased with the product quality. It is outstanding
0 Service was okay but I had higher expectations.
0 I am extremely disappointed with the delayed delivery in this sprint
1 This product is just amazing!
1 I can sympathize with your frustration.
0 This is one of the worst experiences I've ever had in my life.
0 I would be happy to to include this text in my book but I can't do that at this moment.
Next step
Feedback

Figure 6: Add data for your model.

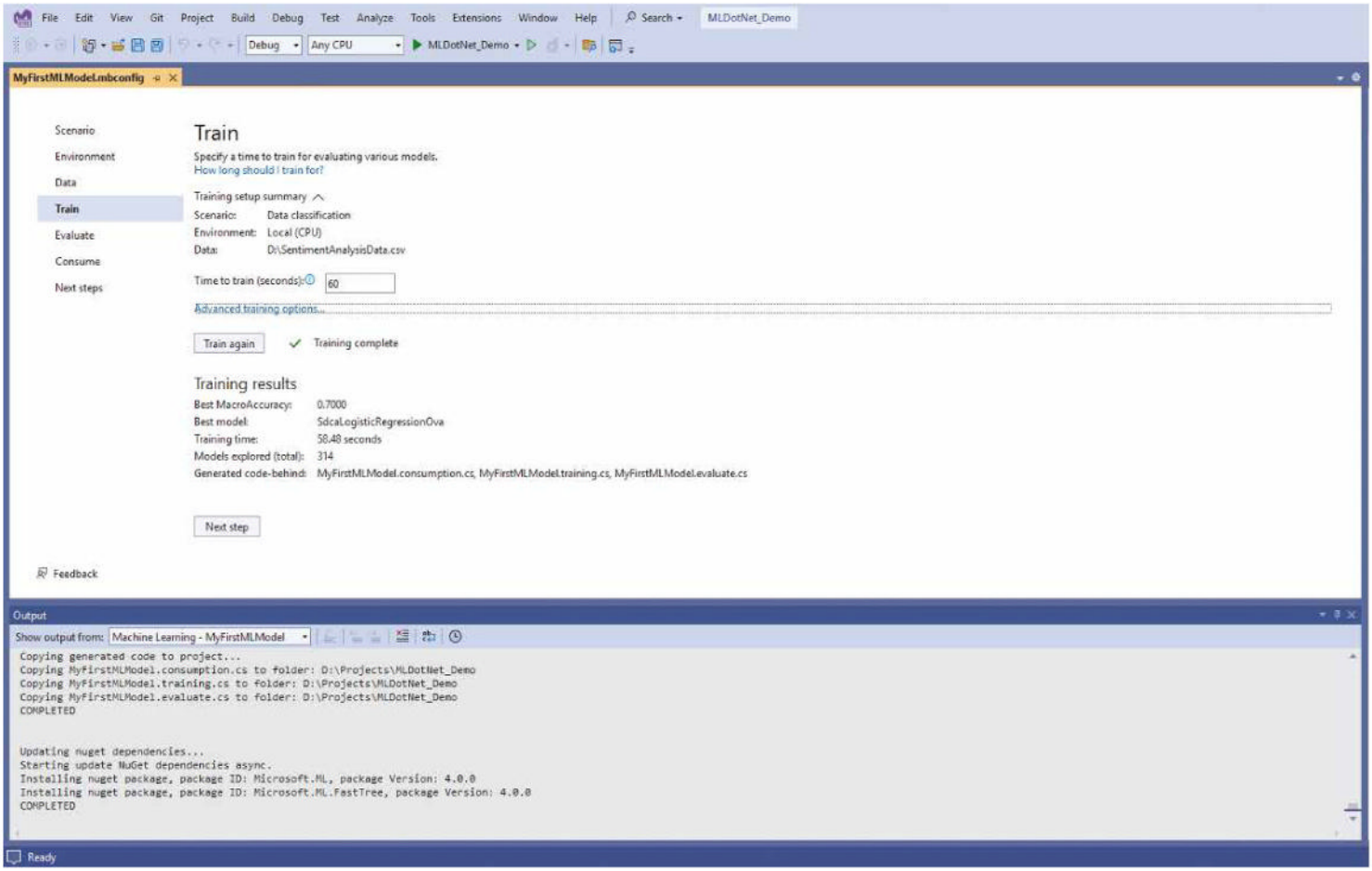


Figure 7: The training results are displayed upon changing the training duration and clicking on the Train again button

The Model Builder also generates sample source code as a reference to consume the model. For example, you can use the following piece of code to load sample data:

```
var sampleData =
new MyFirstMLModel.ModelInput()
{
SentimentText = @"The product is fine
but it did not satisfy my expectations.",
};
```

The following code snippet shows how you can load the model and then invoke the Predict method on the model instance to predict output based on the input data provided:

```
var result = MyFirstMLModel.Predict(sampleData);
```

If you choose to generate your model consumer application using the project in-built templates, the new project (you'll need to choose between a console app or a Web Api app, as shown in **Figure 9**) will be created and added to the solution currently in your Visual Studio IDE. Remember, you can always alter the generated code based on your requirements.

In the sections that follow, I'll examine how to take advantage of ML.NET to predict product prices in an ASP.NET Core application.

Implementing a Product Price Prediction API Using ML.NET and ASP.NET Core

In this section, you'll implement a real-life application that uses machine learning algorithms to predict product prices. To build this real-life application, follow these steps:

1. Create the database.
1. Create an ASP.NET Core Web API project.
2. Add the model to the project.
3. Train and evaluate the model.
4. Retrain the model.
5. Predict the product price.

Figure 10 illustrates the complete flow of how the components of the application work.

Create a New ASP.NET Core 10 Project in Visual Studio 2022

You can create a project in Visual Studio 2022 in several ways, such as from the Visual Studio 2022 Developer Command Prompt or by launching the Visual Studio 2022 IDE. When you launch Visual Studio 2022, you'll see the Start window. You can choose **Continue without code** to launch the main screen of the Visual Studio 2022 IDE.

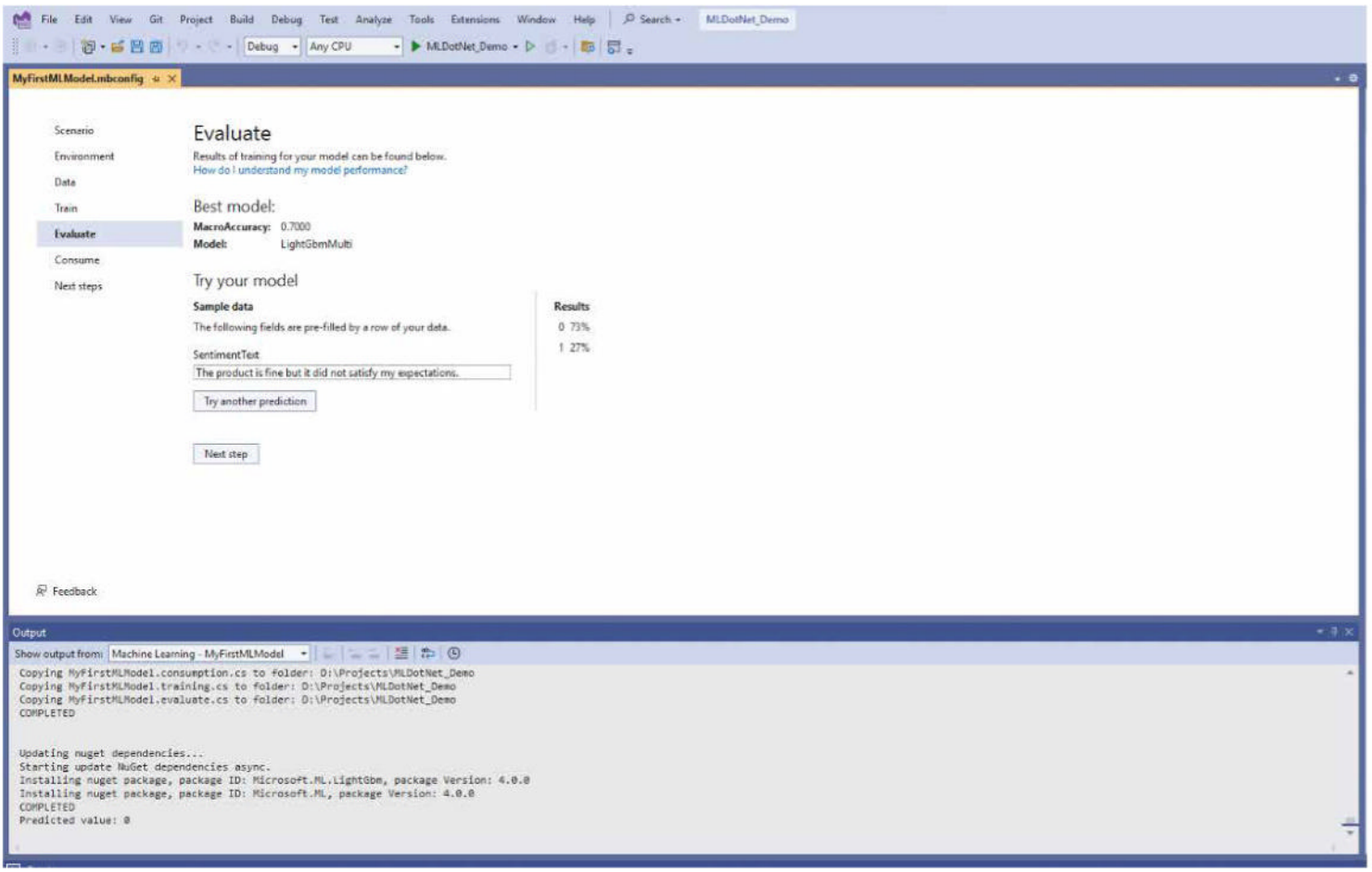


Figure 8: The ML model in action

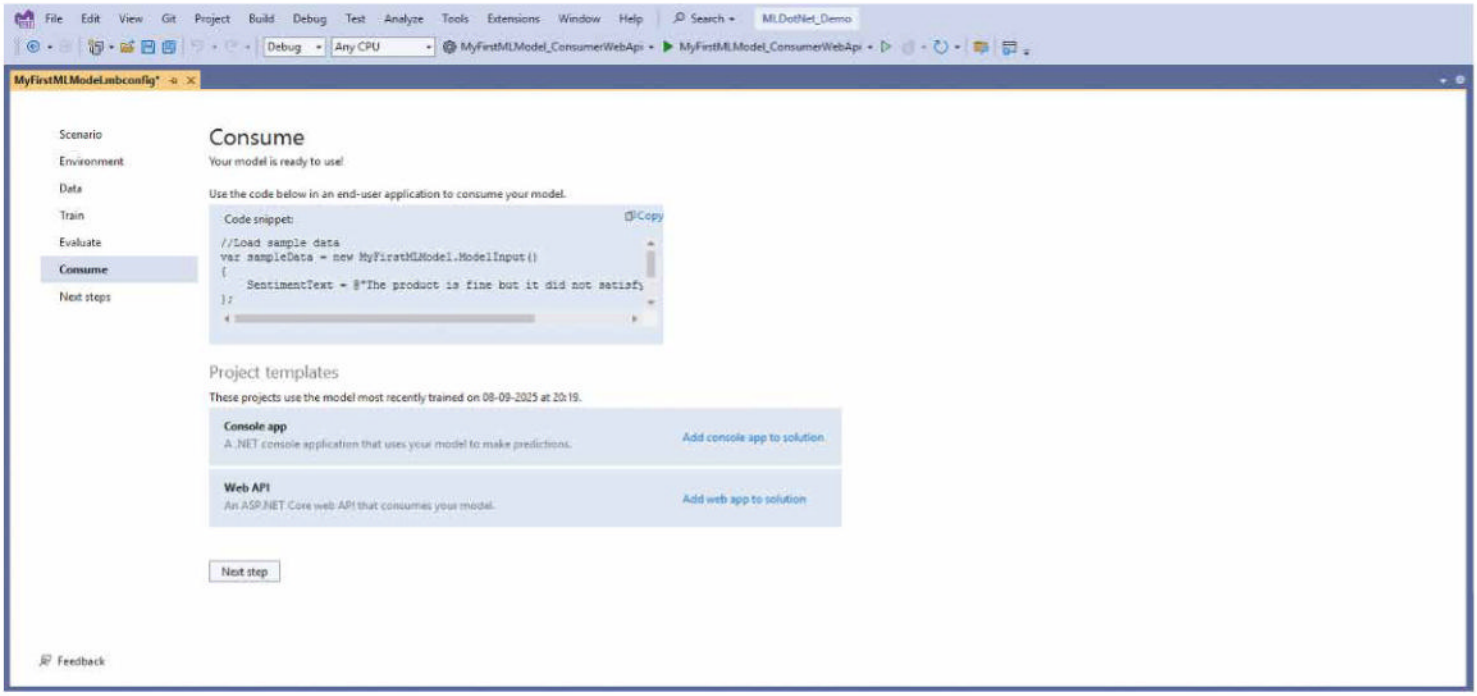


Figure 9: Consuming the model

Now that you know the basics, let's start setting up the project. To create a new ASP.NET Core 8 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 IDE.
2. In the **Create a new project** window, select **ASP.NET Core Web API** and click Next to move on.
3. Specify the project name as **ASPNETCoreMLDemo** and the path where it should be created in the **Configure your new project** window.
4. If you want the solution file and project to be created in the same directory, you can optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
5. In the next screen, specify the target framework and authentication type. Ensure that the **Configure for HTTPS**, **Enable Docker Support**, **Do not use top-level statements**, and the **Enable OpenAPI support** checkboxes are unchecked because you won't use any of these in this example.
6. Remember to leave the **Use controllers** checkbox checked because you won't use minimal API in this example.
7. Click Create to complete the process.

A new ASP.NET Core Web API project is created. You'll use this project to implement the CQRS pattern in ASP.NET Core and C#.

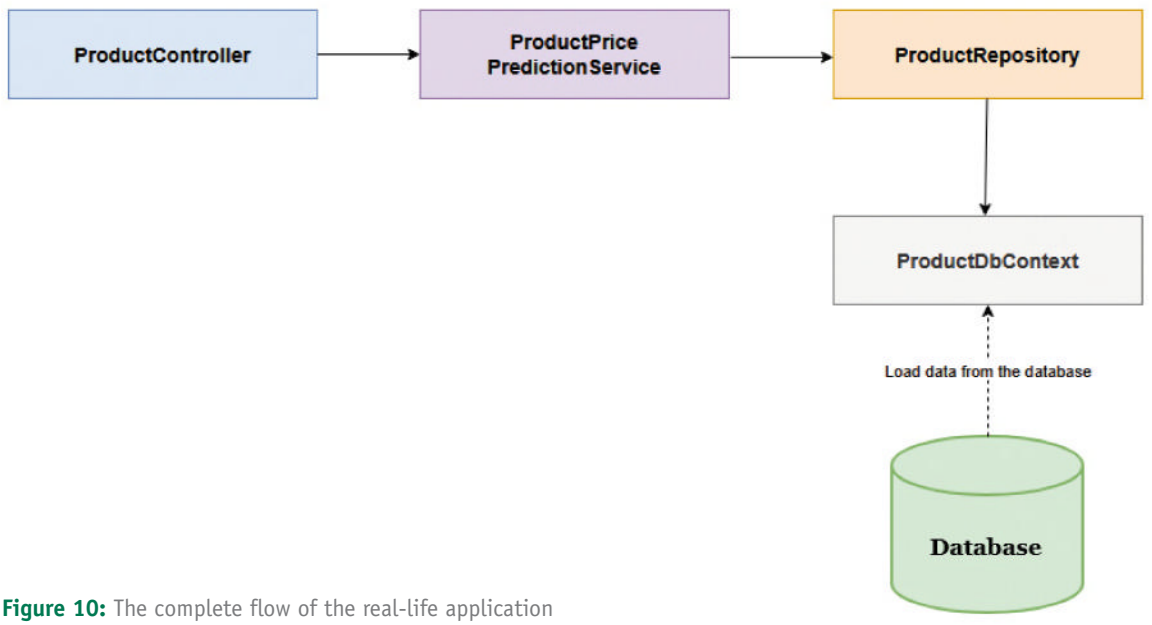


Figure 10: The complete flow of the real-life application

Listing 1: The Database Script

```
-- Drop the Product database table
-- if already it exists
IF OBJECT_ID('dbo.Product', 'U') IS NOT NULL
    DROP TABLE dbo.Product;
GO


-- Create the Product database table
CREATE TABLE dbo.Product
(
    Product_Id INT IDENTITY(1,1) PRIMARY KEY,
    Product_Code VARCHAR(5) NOT NULL,
    Product_Type VARCHAR(25) NOT NULL,
    Region_Code VARCHAR(2) NOT NULL,
    Product_Price DECIMAL(10,2) NOT NULL
);
GO


;WITH Numbers AS
(
    SELECT TOP (100) ROW_NUMBER()
    OVER (ORDER BY (SELECT NULL)) AS N
    FROM sys.all_objects
)
INSERT INTO dbo.Product
(Product_Code, Product_Type,
Region_Code, Product_Price)
SELECT
    -- Map product codes
    CASE ((N-1) % 4)
        WHEN 0 THEN 'P0001'
        WHEN 1 THEN 'P0002'
        WHEN 2 THEN 'P0003'
        WHEN 3 THEN 'P0004'
    END AS Product_Code,


    -- Map product types
    CASE ((N-1) % 4)
        WHEN 0 THEN 'Laptop'
        WHEN 1 THEN 'Furniture'
        WHEN 2 THEN 'Smart Phone'
        WHEN 3 THEN 'Tablet'
    END AS Product_Type,


    'R' + CAST(((N-1) % 4) + 1
AS VARCHAR(1)) AS Region_Code,

    -- Generate the Product price logic
    CASE
        -- P0002 more in R1
        WHEN ((N-1) % 4) = 1
AND ((N-1) % 4) + 1 = 1
THEN 1600 + (N % 50)
        -- P0001 more in R2
        WHEN ((N-1) % 4) = 0
AND ((N-1) % 4) + 1 = 2
THEN 1550 + (N % 40)
        -- P0003 more in R1 or R2
        WHEN ((N-1) % 4) = 2
AND (((N-1) % 4) + 1
IN (1,2)) THEN 1400 + (N % 60)
        -- P0004 more in R3 or R4
        WHEN ((N-1) % 4) = 3
AND (((N-1) % 4) + 1
IN (3,4)) THEN 1350 + (N % 30)
        -- Base price with variation
        ELSE 1200 + (N % 100)
    END AS Price
FROM Numbers;
GO

-- Verify data
      SELECT * FROM dbo.Product
ORDER BY Product_Id;
```

Create the Database

In this example, you'll create a database named StockManagementSystem that contains one table called Product. You'll write a SQL script that generates several product records (about 100 records) with the following fields:

- Product_Id (integer, primary key)
- Product_Code (varchar 5)
- Product_Type (varchar 25)
- Region_Code (varchar 2)
- Price (decimal)
- Stock_Quantity (integer)

The records in the Product database table will be generated based on the following assumptions:

- There cannot be two records having the same combination of region code, product code, and price.
- The price of the product with product code P0002 will be more if the region code is R1.
- The price of the product P0001 will be more if the region code is R2.
- The price of the product P0003 will be more if the region code is R1 or R2.
- The price of the product P0004 will be more if the region code is R3 or R4.
- There will only be four region codes: R1, R2, R3, and R4. Typical product codes will be P0001, P0002, P0003, and P0004.
- The same product should be available in multiple regions, i.e., P001 should be available in R2, R3, and R4.
- The price of a product will vary in different regions, i.e., if the region codes change, the product price will also change.
- As an example, a laptop can have a price of 1500 in R2 but it can be 1450 in R3 and 1475 in R4, and so on.
- Typical product types can be Laptop, Furniture, Smart Phone, etc.
- Each product type will have a distinct product code.

The following code snippet shows how you can create a database table named Products.

```
CREATE TABLE dbo.Products(
    Product_Id INT IDENTITY(1, 1) PRIMARY KEY,
    Product_Code VARCHAR(5) NOT NULL,
    Product_Type VARCHAR(25) NOT NULL,
    Region_Code VARCHAR(2) NOT NULL,
    Product_Price DECIMAL(10, 2) NOT NULL,
    Stock_Quantity INT NOT NULL
);
```

The following piece of code shows how you can define a unique constraint so that no two records in this table

have the same combination of Product_Code, Region_Code, and Product_Price.

```
ALTER TABLE dbo.Products
ADD CONSTRAINT
UQ_ProductRegionPrice
UNIQUE(Product_Code,
Region_Code, Product_Price);
GO
GO
```

The complete script for creating the database is given in **Listing 1**.

Install the NuGet Packages

The next step is to install the necessary NuGet Package(s) for the application. In this application, you'll need EF Core and ML.NET. To install the required packages into your project, right-click on the solution and then select **Manage NuGet Packages for Solution....**

Once the window pops up, search for the NuGet packages to add to your project. To do this, type in Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Design, Microsoft.EntityFrameworkCore.Tools, Microsoft.EntityFrameworkCore.SqlServer, and Microsoft.ML in the search box and install them one after the other. Alternatively, you can type the commands shown below at the NuGet Package Manager Command Prompt:

```
PM> Install-Package
Microsoft.EntityFrameworkCore

PM> Install-Package
Microsoft.EntityFrameworkCore.Design

PM> Install-Package
Microsoft.EntityFrameworkCore.Tools

PM> Install-Package
Microsoft.EntityFrameworkCore.SqlServer

PM> Install-Package
Microsoft.ML
```

Create the Model Classes

Create a new class called Product in a file named Product.cs inside the Models folder and write the following code in there:

```
public class Product
{
    public int Product_Id
    {
        get;
        set;
    }


    public string Product_Code
    {
        get;
        set;
    } = null!;

    public string Product_Type
    {
        get;
        set;
    } = null!;


    public string Region_Code
    {
        get;
        set;
    } = null!;


    public decimal Product_Price
    {
        get;
        set;
    }


    public int Stock_Quantity
    {
        get;
        set;
    }
}
```

You'll create two more model classes, ProductPriceInput and ProductPricePrediction classes, as shown below.

```
public class ProductPriceInput
    {
        public string Product_Code
{ get; set; }
        public string Region_Code
{ get; set; }
        public float Product_Price
```

Listing 2: The ProductDbContext class

```
public class ProductDbContext : DbContext
    {
        private readonly IConfiguration _configuration;

        public ProductDbContext(DbContextOptions
          <ProductDbContext>  options,
          IConfiguration configuration) : base(options)
        {
            _configuration = configuration;
        }

        protected override void
         OnConfiguring(DbContextOptionsBuilder
          optionsBuilder)
        {
            _ = optionsBuilder.UseSqlServer(_configuration.

              GetConnectionString("DefaultConnection")).
              EnableSensitiveDataLogging();
        }

        public DbSet<Product> Products { get; set; }
        protected override void OnModelCreating
        (ModelBuilder modelBuilder)
        {

            modelBuilder.Entity<Product>().ToTable("Product");
            modelBuilder.Entity<Product>().HasKey(p =>
              p.Product_Id);
        }
}
```

```
{ get; set; }
    }

    public class ProductPriceOutput
    {
        public float Product_Price
{ get; set; }
        public float Score
{ get; set; }
    }
```

ProductPriceInput will be used to store a combination of region code and product code data and pass it as a parameter to the HttpPost method in the ProductController, and ProductPricePrediction will be used to store the result of the prediction.

Create the Data Context

In Entity Framework Core (EF Core), a data context is a component used by an application to interact with the database and manage database connections, and to query and persist data in the database. Let's now create the data context class to enable the application to interact with the database to perform CRUD (Create, Read, Update, and Delete) operations.

To do this, create a new class named ProductDbContext that extends the DbContext class of EF Core and write the following code in there:

```
public class ProductDbContext:
DbContext
{
    public DbSet <Product>
    Products { get; set; }

    protected override void OnConfiguring
    (DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring (optionsBuilder);
    }
}
```

In the preceding piece of code, the statement base.OnConfiguring(optionsBuilder) calls the OnConfiguring method of the base class of your ProductDbContext. You can specify your database connection string in the OnConfiguring overloaded method of the ProductDbContext class. However, in this implementation, you'll store the database connection settings in the AppSettings.json file and read it in the Program.cs file to establish a database connection. The following code snippet illustrates how you can define a public constructor for your data context class.

```
public ProductDbContext
(DbContextOptions
<ProductDbContext> options,
IConfiguration configuration) :
base(options)
{
   _configuration = configuration;
}
```

Register the ProductDb data context instance as a service with the services container of ASP.NET Core using the following piece of code in the Program.cs file:

```
builder.Services.AddDbContext
<ProductDbContext>(options =>
{
    options.UseSqlServer(
      builder.Configuration
["ConnectionStrings:DefaultConnection"]);
});
```

If your application needs to perform multiple units of work, it's advisable to use a DbContext factory instead. To do this, register a factory by calling the AddDbContext-Factory method in the Program.cs file of your project, as shown in the following code example:

```
builder.Services.AddDbContextFactory
<ProductDbContext>(options =>
{
    options.UseSqlServer
(builder.Configuration
["ConnectionStrings: DefaultConnection"]);
});
```

The complete source code of ProductDbContext class is given in **Listing 2**.

Create the ProductRepository Class

A repository class is an implementation of the Repository design pattern and is one that manages data access. The application takes advantage of the repository instance to perform CRUD operations against the database. In this example, you'll use the repository to load data only.

Listing 3: The ProductRepository class

```
public class ProductRepository :
IProductRepository
{
    private readonly ProductDbContext _productDbContext;
    public ProductRepository
(ProductDbContext productDbContext)
    {
        _productDbContext = productDbContext;
    }

    public async Task<List<Product>>
GetAllProductsAsync()
    {
        return await
_productDbContext.Products.ToListAsync();
    }


    public async Task<Product>
GetProductByIdAsync(int id)
    {
        return await _productDbContext.Products.
        FirstOrDefaultAsync
(p => p.Product_Id == id);
    }


    public async Task<List<Product>>
GetProductsByCodeAndRegionAsync
(string productCode, string regionCode)
    {
        return await _productDbContext.Products
            .Where(p => p.Product_Code ==
productCode && p.Region_Code == regionCode)
            .ToListAsync();
    }
}
```

Create a new interface named IProductRepository in a file having the same name and write the following code in there:

```
public interface IProductRepository
{


}
```

Now, create a new class named ProductRepository in a file having the same name with a .cs extension. The ProductRepository class should extend the IProductRepository interface and implement its members.

```
public class ProductRepository :
IProductRepository
{

}
```

Here is how the IProductRepository interface should look with all its members in there:

```
public interface IProductRepository
{
   public Task<List<Product>>
GetAllProductsAsync();
   public Task<Product>
GetProductByIdAsync(int id);
   public Task<List<Product>>
   GetProductsByCodeAndRegionAsync
      (string productCode,
string regionCode);
}
```

The ProductRepository class, illustrated in the code snippet below, implements the methods of the IProductRepository interface.

```
public async Task<List<Product>>
GetAllProductsAsync()
{
    return await await
_productDbContext.Products.
ToListAsync();
}


public async Task<Product>
GetProductByIdAsync(int id)
{
    return
await _productDbContext.Products.
FirstOrDefaultAsync
    (p => p.Product_Id == id);
}


public async Task<List<Product>>
        GetProductsByCodeAndRegionAsync
(string productCode, string regionCode)
```

Listing 4: The ProductPricePredictionService class

```
public class
ProductPricePredictionService :
IProductPricePredictionService
{
    private readonly
IProductRepository _productRepository;
    public ProductPricePredictionService
(IProductRepository repository)
    {
        _productRepository = repository;
    }


    public async Task<double> PredictPrice
(string productCode, string regionCode)
    {
        var mlContext = new MLContext();
        var products = await
_productRepository.GetAllProductsAsync();
        var data = products.Select(p =>
new ProductPriceInput
        {
            Product_Code = p.Product_Code,
            Region_Code = p.Region_Code,
            Product_Price =
Convert.ToSingle(p.Product_Price)
        }).ToList();


  var dataView =
mlContext.Data.LoadFromEnumerable(data);

  var pipeline = mlContext.Transforms.Text.
FeaturizeText(inputColumnName:
@"Product_Code", outputColumnName: @"Product_Code")
                .Append(mlContext.Transforms.Text.
FeaturizeText(inputColumnName: @"Region_Code",
outputColumnName: @"Region_Code"))
                .Append(mlContext.Transforms.Concatenate
(@"Features", new[]
{ @"Product_Code", @"Region_Code" })
.Append(mlContext.Transforms.
CopyColumns("Label", "Product_Price"))
.Append(mlContext.Regression.Trainers.Sdca()));


        var model = pipeline.Fit(dataView);


        PredictionEngine<ProductPriceInput,
ProductPriceOutput> _predictionEngine =
mlContext.Model.CreatePredictionEngine
<ProductPriceInput, ProductPriceOutput>(model);


        ProductPriceInput sampleData =
new ProductPriceInput()
        {
            Product_Code = productCode,
            Region_Code = regionCode
        };


        var predictionResult = _predictionEngine.Predict(sampleData);


        return await
Task.FromResult(
Math.Round(predictionResult.Score, 2));
    }
}
```

```
{
    return
await _productDbContext.Products
          .Where(p =>
p.Product_Code ==
    productCode &&
p.Region_Code == regionCode)
          .ToListAsync();
}
```

The complete source code of the ProductRepository class is given in **Listing 3.**

Create the ProductPricePredictionService Class

Next, create a class named ProductPricePredictionService that will contain the actual logic for leveraging ML.NET to predict the product price. To do this, create a new class named ProductPricePredictionService in a file named ProductPricePredictionService.cs and replace the generated code using the code snippet given below:

```
public class
ProductPricePredictionService :
IProductPricePredictionService
{
    private readonly
IProductRepository _productRepository;
    public ProductPricePredictionService
(IProductRepository repository)
    {
        _productRepository = repository;
    }

    public async Task<double>
PredictPrice(string productCode,
string regionCode)
    {
       //Not yet implemented
    }
}
```

In the preceding code snippet, an instance of type IProductRepository is passed to the constructor of the ProductPricePredictionService class using constructor injection. The ProductPricePredictionService class will be used by the ProductController class to return prediction results.

You now need to load the data from the database, transform the data, and create a prediction engine instance to predict data based on input values. The following code snippet shows how this can be achieved.

```
public async Task<double>
PredictPrice(string productCode,
string regionCode)
{
var mlContext = new MLContext();
var products = await
 _productRepository.GetAllProductsAsync();
var data = products.
Select(p => new ProductPriceInput
{
   Product_Code = p.Product_Code,
   Region_Code = p.Region_Code,
   Product_Price =
 Convert.ToSingle
(p.Product_Price)
  }).ToList();


var dataView =
mlContext.Data.
LoadFromEnumerable(data);


var pipeline =
mlContext.Transforms.Text.
FeaturizeText(inputColumnName:
@"Product_Code", outputColumnName:
@"Product_Code")
                        .Append(mlContext.Transforms.Text.
FeaturizeText(inputColumnName:
@"Region_Code", outputColumnName:
@"Region_Code"))
                        .Append(mlContext.Transforms.
Concatenate(@"Features", new[]
{ @"Product_Code", @"Region_Code" })
                        .Append(mlContext.Transforms.
CopyColumns("Label", "Product_Price"))
                        .Append(mlContext.Regression.
Trainers.Sdca()));

var model = pipeline.Fit(dataView);
```

Listing 5: The Program.cs file

```
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
var connectionString =
builder.Configuration.
GetConnectionString("DefaultConnection");


builder.Services.AddDbContext
<ProductDbContext>(options =>
{
    options.UseSqlServer(
      builder.Configuration
["ConnectionStrings:DefaultConnection"]);
});


builder.Services.AddScoped<IProductRepository,
ProductRepository>();
builder.Services.AddScoped
<IProductPricePredictionService,
ProductPricePredictionService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseHttpsRedirection();

//app.UseAuthorization();

app.MapControllers();

      app.Run();
```

Listing 6: The Product.razor page

```
@page "/products"
@inject HttpClient Http


<h3>Product List</h3>
@if (products == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Id</th>
                <th>Code</th>
                <th>Name</th>
                <th>Price</th>
                <th>Stock</th>
                <th>Region</th>
            </tr>
        </thead>
        <tbody>
        @foreach (var prod in products)
        {
            <tr>
                <td>@prod.Id</td>
                <td>@prod.Code</td>
                <td>@prod.Name</td>
                <td>@prod.Price</td>
                <td>@prod.Stock</td>
                <td>@prod.RegionCode</td>
            </tr>
        }
        </tbody>
    </table>
}


<h4>Predict Price</h4>
<div>
    <label>Product Code: </label>
    <input @bind="input.Code"
style="margin-right:2em;" />
    <label>Region Code: </label>
    <input @bind="input.RegionCode"
style="margin-right:2em;" />
    <button class="btn btn-primary" @onclick="PredictPriceAsync">Predict</
button>
</div>

@if (predictedPrice != null)
{
    <div style="margin-top:1em;">
        <strong>Predicted Price:
</strong>@predictedPrice
    </div>
}


@code {
    private readonly string
apiBase = "https://localhost:5001";
    List<Product> products;
    PredictInput input = new();
    string predictedPrice;


    protected override async
Task OnInitializedAsync()
    {
        products = await Http.GetFromJsonAsync<List
<Product>>($"{apiBase}/api/products");
    }


    async Task PredictPriceAsync()
    {
        var response =
await Http.PostAsJsonAsync(
$"{apiBase}/api/products/predict", input);
        if (response.IsSuccessStatusCode)
            predictedPrice = await response.Content.ReadAsStringAsync();
        else
            predictedPrice = "Error";
    }


    public class Product
    {
        public int Id { get; set; }
        public string Code { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
        public int Stock { get; set; }
        public string RegionCode { get; set; }
    }
    public class PredictInput
    {
        public string Code { get; set; }
        public string RegionCode { get; set; }
    }
}
```

```
PredictionEngine<ProductPriceInput,
ProductPriceOutput>
_predictionEngine = mlContext.Model.
CreatePredictionEngine
<ProductPriceInput,
ProductPriceOutput>(model);


ProductPriceInput sampleData =
new ProductPriceInput()
{
  Product_Code = productCode,
  Region_Code = regionCode
};
```

```
var predictionResult =
_predictionEngine.Predict
(sampleData);


return await Task.FromResult
(Math.Round(predictionResult.
Score, 2));
}
```

The complete source code of the ProductPricePrediction-Service class is given in **Listing 4**.

Register the Services

To be able to leverage dependency injection to access instances of the repository and the service class you created earlier, register these instances with the service

Listing 7: The ProductController class

```
   [ApiController]
   [Route("api/[controller]")]
    public class ProductController :
ControllerBase
    {
        private readonly
IProductPricePredictionService
_productPricePredictionService;
        public ProductController
(IProductPricePredictionService
productPricePredictionService)
        {
            _productPricePredictionService =
productPricePredictionService;
        }


        [HttpPost("predict")]
        public async Task<ActionResult<double>>
        Predict([FromBody]
ProductPriceInput productPriceInput)
        {
            if (productPriceInput == null ||
string.IsNullOrWhiteSpace(
productPriceInput.Product_Code) ||
          string.IsNullOrWhiteSpace
(productPriceInput.Region_Code))
                return BadRequest
("Product_Code and Region_Code required");
            var price =
await _productPricePredictionService.PredictPrice
                (productPriceInput.Product_Code,
productPriceInput.Region_Code);
            return Ok(price);
        }
 }
```

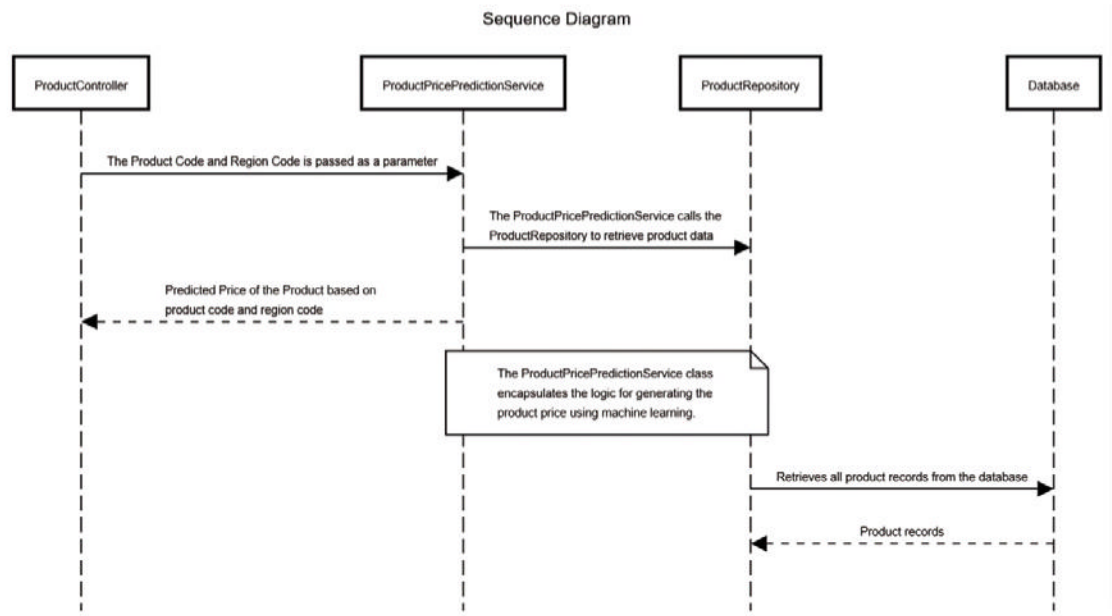

Sequence Diagram
ProductController
ProductPricePredictionService
ProductRepository
Database
The Product Code and Region Code is passed as a parameter
The ProductPricePredictionService calls the ProductRepository to retrieve product data
Predicted Price of the Product based on product code and region code
The ProductPricePredictionService class encapsulates the logic for generating the product price using machine learning.
Retrieves all product records from the database
Product records

Figure 11: Sequence diagram of the complete flow

collection by writing the following piece of code in the Program.cs file:

```
builder.Services.AddScoped
<IProductRepository, ProductRepository>();
builder.Services.AddScoped
<IProductPricePredictionService, ProductPricePredictionService>();
```

Also register the ProductDbContext with the service collection so that you can access an instance of the ProductDbContext class in your product repository class or other classes if needed.

```
builder.Services.AddDbContext
<ProductDbContext>(options =>
{
    options.UseSqlServer(
      builder.Configuration
["ConnectionStrings:
DefaultConnection"]);
});
```

The complete source code of the Program.cs file is given in **Listing 5**.

Create the ProductController Class

Now, create a new controller named ProductController in the Controllers folder of the project. The following code snippet shows how you can take advantage of constructor injection to pass an instance of type IProductPrediction-Service interface using the constructor of the Product-Controller class.

```
public class ProductController :
ControllerBase
{
private readonly
IProductPricePredictionService
_productPricePredictionService;


public ProductController
(IProductPricePredictionService
```

```
IProductPricePredictionService
productPricePredictionService)
{
    _productPricePredictionService =
    productPricePredictionService;
}
    //Action methods go here
}
```

Now, you'll create an HTTP POST endpoint that will be used to predict the price of a product if product code and region code is given to it as parameters. The following code snippet shows how you can create an action method that accepts product code and region code as parameters and calls the PredictPrice method of the ProductPricePredictionService class to get the predicted price of the product.

```
[HttpPost("predict")]
public async Task
<ActionResult<double>>
Predict([FromBody]
ProductPriceInput productPriceInput)
{
  if (productPriceInput == null ||
  string.IsNullOrWhiteSpace
(productPriceInput.Product_Code) ||
string.IsNullOrWhiteSpace
(productPriceInput.Region_Code))
return BadRequest
("Product_Code and
Region_Code are required");


var price =
await _productPricePredictionService.
PredictPrice
 (productPriceInput.Product_Code,
productPriceInput.Region_Code);
  return Ok(price);
}
```

The complete source code of the ProductController class is given in **Listing 7**. The sequence diagram of the complete flow is given in **Figure 11**.

Create the User Interface in Blazor

In this section, you'll take advantage of Blazor to create the user interface that will consume the API methods you created earlier. Blazor, an open-source framework from Microsoft, empowers developers to create interactive web applications using C# and .NET. Blazor employs WebAssembly, a web standard that permits browser-based code execution from languages other than JavaScript, creating a fully featured, high-performance client-side development environment using C#. Blazor provides a modern, efficient, and versatile approach to web application development. The complete source code of the user interface is given in **Listing 6**.

Create a New Blazor Web Assembly Project in .NET 10 and Visual Studio 2022

You can create a project in Visual Studio 2022 in several ways, such as from the Visual Studio 2022 Developer Command Prompt or by launching the Visual Studio 2022 IDE. When you launch Visual Studio 2022, you'll see the Start window. You can choose "Continue without code" to launch the main screen of the Visual Studio 2022 IDE.

Now that you know the basics, let's start setting up the project. To create a new ASP.NET Core 8 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 IDE.
2. In the **Create a new project** window, select **Blazor Web App** and click Next to move on.
3. Specify the project name as **Product_Price_UI** and the path where it should be created in the **Configure your new project** window.
4. If you want the solution file and project to be created in the same directory, you can optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
5. In the next screen, specify the target framework and authentication type as well. Ensure that the **Configure for HTTPS**, and **Do not use top-level statements** checkboxes are unchecked because you won't use any of these in this example.
6. Next, specify the **Interactive render mode** and **Interactivity location.**
7. You should ensure that the **Include sample pages** checkbox is checked if you would like to have sample pages added to your project
8. Click **Create** to complete the process.

Implementing a Sales Forecast API Using ML.NET and ASP.NET Core

In this section, you'll implement an API that forecasts sales data of products based on historical data (i.e., sales data for the last ten years).

Create the Models

In this implementation, you'll use the following model classes:

- **ProductSales:** Contains the sales value of a product
- **ProductSalesRecord:** Stores forecasted sales data
- **SalesForecastInput:** Stores input data
- **SalesForecastOutput**: Stores sales forecast output data
- **SalesForecast**: Contains a float array used to calculate sales forecast data.

The following piece of code shows how these model classes should look:

```
namespace SalesForecastAPI.Models
{
    public class ProductSales
    {
public float Sales
{ get; set; }
    }
}

namespace SalesForecastAPI.Models
{
public class ProductSalesRecord
{
public string Product_Code
```

```
{ get; set; }
        public int Year
{ get; set; }
        public int Month
{ get; set; }
        public int Sales
{ get; set; }
    }
}

namespace SalesForecastAPI.Models
{
    public class SalesForecastInput
    {
        public string
Product_Code { get; set; }
        public int Month
{ get; set; }
    }
}

namespace SalesForecastAPI.Models
{
    public class SalesForecastOutput
    {
        public int Month
{ get; set; }
        public int Actual
{ get; set; }
        public int Forecast
{ get; set; }
    }
}

namespace SalesForecastAPI.Models
{
    public class SalesForecast
    {
        public float[]
ForecastedSales { get; set; }
    }
}
```

Create the SalesTimeSeriesForecastService Class

Now, create a new interface named ISalesTimeSeriesForecastService in a file named ISalesTimeSeriesForecastService.cs and replace the default generated code with the following piece of code.

```
using SalesForecastAPI.Models;

namespace SalesForecastAPI
{
    public interface
ISalesTimeSeriesForecastService
    {
        Task<List<
SalesForecastOutput>>
GetSalesForcastData
(SalesForecastInput
salesForecastInput);
    }
}
```

Next, create a new class named SalesTimeSeriesForecastService that implements the ISalesTimeSeriesForecastService interface in a file having an identical name with a .cs extension. In this class, you'll generate sales data that will be used to forecast sales of products. Apart from generating sales data, this class will also contain the necessary logic for generating forecasted sales data.

For the sake of simplicity, you'll generate a list of records instead of reading them from the database. The following piece of code shows how you can generate sales data for ten years for five products.

```
private List
<ProductSalesRecord>
GenerateSalesData()
{
  List <ProductSalesRecord> salesData =
   new List <ProductSalesRecord> ();


  string[] productCodes = {
    "P0001",
    "P0002",
    "P0003",
    "P0004",
    "P0005"
  };

  Random rand = new Random();

  for (int year = 2016; year <= 2025; year++)
  {
    for (int month = 1; month <= 12; month++)
    {
      foreach(var code in productCodes)
      {
        float baseSales =
        110 f + 13 f *
Array.IndexOf(productCodes, code);
        float season =
        18 f * (float)
Math.Sin(2 * Math.PI * month / 12);
        float trend =
(year - 2016) * 2.5 f;
        float noise =
(float)(rand.NextDouble() * 6 - 3);


        float sales =
        (float)(Math.Round(
        baseSales + season + trend + noise));

        salesData.Add(new ProductSalesRecord
        {
          Product_Code = code,
            Year = year,
            Month = month,
            Sales = sales
        });
      }
    }
  }


  return salesData;
}
```

The GenerateSalesForecast method given below generates sales data using a time series prediction engine. Note how the CreateTimeSeriesEngine method has been called.

```
private float[]
GenerateSalesForecast
(string productCode,
int forecastHorizon,
List<ProductSalesRecord> salesRecords)
{
    var series = salesRecords
        .Where(r =>
r.Product_Code == productCode)
        .OrderBy(r =>
r.Year * 100 + r.Month)
        .Select(r =>
new ProductSales {
Sales = (float)r.Sales })
        .ToList();


    int seriesLength = series.Count;
    int windowSize =
Math.Max(2,
Math.Min(12, seriesLength / 20));
    if (seriesLength <= 2 * windowSize)
        return null;


    var dataView =
_mlContext.Data.LoadFromEnumerable(series);


    var pipeline =
_mlContext.Forecasting.ForecastBySsa(
        outputColumnName:
nameof(SalesForecast.ForecastedSales),
        inputColumnName:
nameof(ProductSales.Sales),
        windowSize: windowSize,
        seriesLength: seriesLength,
        trainSize: seriesLength,
        horizon: forecastHorizon
    );
    var model = pipeline.Fit(dataView);
    var engine =
model.CreateTimeSeriesEngine
<ProductSales, SalesForecast>(_mlContext);


    return engine.Predict().ForecastedSales;
}
```

In the GetSalesForecastData method, the GenerateSalesForecastData method is called. This method returns a list of float values corresponding to the sales of products for one or more months.

```
var productSalesRecords =
GenerateSalesData();
string prodCode =
salesForecastInput.Product_Code;
int userMonth = int.Parse
(salesForecastInput.
Month.ToString());


int latestYear =
productSalesRecords.Max
(r => r.Year);
var yearData =
productSalesRecords.
Where(r => r.Product_Code ==
prodCode && r.Year == latestYear)
.OrderBy(r => r.Month).ToList();
float[] forecasts =
GenerateSalesForecast
(prodCode, 12,
productSalesRecords);
```

Next, this list is used to create a formatted list of sales forecast data using the SalesForecastOutput class. The complete source code for the SalesTimeSeriesForecastService class is given in **Listing 8.** The complete source code of the GetSalesForecastData method is given in **Listing 9.**

Listing 8: The SalesTimeSeriesForecastService class

```
using Microsoft.ML;
using Microsoft.ML.Transforms.TimeSeries;
using SalesForecastAPI.Models;

namespace SalesForecastAPI
{
    public class
SalesTimeSeriesForecastService :
ISalesTimeSeriesForecastService
    {
        private readonly MLContext _mlContext =
new MLContext();
        private List<ProductSalesRecord>
GenerateSalesData()
        {
            List<ProductSalesRecord>
salesData = new List<ProductSalesRecord>();
            string[] productCodes = {
"P0001", "P0002", "P0003", "P0004", "P0005" };
            Random rand = new Random();


            for (int year = 2016;
year <= 2025; year++)
            {
                for (int month = 1;
month <= 12; month++)
                {
                    foreach (var code
in productCodes)
                    {
                        float baseSales =
110f + 13f * Array.IndexOf(productCodes, code);
                        float season =
18f * (float)Math.Sin(2 * Math.PI * month / 12);
                        float trend =
(year - 2016) * 2.5f;
                        float noise =
(float)(rand.NextDouble() * 6 - 3);


                        int sales =
Convert.ToInt32(Math.Round(
baseSales + season + trend + noise));
```

Listing 8: continued...

```
                        salesData.Add(
new ProductSalesRecord
                        {
                            Product_Code = code,
                            Year = year,
                            Month = month,
                            Sales = sales
                        });
                    }
                }
            }

            return salesData;
        }

        private float[] Forecast(
string productCode,
int forecastHorizon,
List<ProductSalesRecord> salesRecords)
        {
            var series = salesRecords
                .Where(r =>
r.Product_Code == productCode)
                .OrderBy(
r => r.Year * 100 + r.Month)
                .Select(
r => new ProductSales
{ Sales = (float)r.Sales })
                .ToList();

            int seriesLength = series.Count;
            int windowSize = Math.Max(2,
Math.Min(12, seriesLength / 20));
            if (seriesLength <= 2 * windowSize)
                return null;

            var dataView =
 _mlContext.Data.LoadFromEnumerable(series);

            var pipeline =
_mlContext.Forecasting.ForecastBySsa(
                outputColumnName: "ForecastedSales",
                inputColumnName: nameof(ProductSales.Sales),
                windowSize: windowSize,
                seriesLength: seriesLength,
                trainSize: seriesLength,
                horizon: forecastHorizon
            );
            var model = pipeline.Fit(dataView);
            var engine = model.CreateTimeSeriesEngine
<ProductSales, SalesForecastOutput>(_mlContext);

            return engine.Predict().ForecastedSales;
        }

        public List<SalesForecast>
GetSalesForcastData(SalesForecastInput
salesForecastInput)
        {
            var productSalesRecords =
GenerateSalesData();
            string prodCode =
salesForecastInput.Product_Code;
            int userMonth =
int.Parse(salesForecastInput.Month.ToString());

            int latestYear =

productSalesRecords.Max(r => r.Year);
            var yearData =
productSalesRecords.Where
(r => r.Product_Code == prodCode &&
r.Year == latestYear)
.OrderBy(r => r.Month).ToList();

            float[] forecasts =
Forecast(prodCode, 12, productSalesRecords);

            List<SalesForecast>
salesForecasts = new List<SalesForecast>();

            if (userMonth == 0)
            {
                for (int i = 0; i < 12; i++)
                {
                    string actual =
yearData.Count > i ?
yearData[i].Sales.ToString() : "-";
                    string forecast =
(forecasts != null && i < forecasts.Length) ? Math.Round(forecasts[i]).
ToString() : "-";

Console.WriteLine($"{i + 1,5}
{actual,10} {forecast,12}");

                    SalesForecast salesForecast =
 new SalesForecast();
                    salesForecast.Actual =
yearData.Count > i ? yearData[i].Sales : 0;
                    salesForecast.Forecast =
(forecasts != null && i < forecasts.Length) ?
forecasts[i] : 0;
                    salesForecast.Forecast =
Math.Round(salesForecast.Forecast, 2);
                    salesForecast.Month = i + 1;
                    salesForecasts.Add(salesForecast);
                }
            }
            else
            {
                int idx = userMonth - 1;
                string actual =
yearData.Count > idx ?
yearData[idx].Sales.ToString() : "-";
                string forecast =
(forecasts != null && idx < forecasts.Length) ? Math.Round(forecasts[idx]).
ToString() : "-";
                Console.WriteLine($"{userMonth,5}
{actual,10} {forecast,12}");

                SalesForecast salesForecast =
new SalesForecast();
                salesForecast.Actual =
(forecasts != null && idx < forecasts.Length) ?
forecasts[idx] : 0;
                salesForecast.Forecast =
(forecasts != null && idx < forecasts.Length) ?
forecasts[idx] : 0;
                salesForecast.Forecast =
Math.Round(salesForecast.Forecast, 2);
                salesForecast.Month = userMonth;
                salesForecasts.Add(salesForecast);
            }

            return salesForecasts;
        }
    }
}
```

Listing 9: The GetSalesForcastData method

```
public Task<List
<SalesForecastOutput>>
 GetSalesForcastData(SalesForecastInput
salesForecastInput)
{
    var productSalesRecords = GenerateSalesData();
    string prodCode =
salesForecastInput.Product_Code;
    int userMonth =
    int.Parse(salesForecastInput.Month.ToString());


    int latestYear =
productSalesRecords.Max(r => r.Year);
    var yearData = productSalesRecords.Where
(r => r.Product_Code ==
prodCode && r.Year == latestYear)
              .OrderBy(r => r.Month).ToList();


    float[] forecasts = GenerateSalesForecast
(prodCode, 12, productSalesRecords);


    List<SalesForecastOutput>
salesForecasts =
new List<SalesForecastOutput>();


    if (userMonth == 0)
    {
        for (int i = 0; i < 12; i++)
        {
            string actual =
yearData.Count > i ?
yearData[i].Sales.ToString() : "-";
            string forecast =
(forecasts != null && i < forecasts.Length) ?
Math.Round(forecasts[i]).ToString() : "-";
            Console.WriteLine($"{i + 1,5}
{actual,10}
{forecast,12}");


            SalesForecastOutput
salesForecastResult =
new SalesForecastOutput();
            salesForecastResult.Actual =
(int)Math.Ceiling(yearData.Count > i ?
yearData[i].Sales : 0);
            salesForecastResult.Forecast =
(int)Math.Ceiling((forecasts != null &&
i < forecasts.Length) ? forecasts[i] : 0);


            salesForecastResult.Month = i + 1;
            salesForecasts.Add(salesForecastResult);
        }
    }
    else
    {
        int idx = userMonth - 1;
        string actual =
yearData.Count > idx ?
yearData[idx].Sales.ToString() : "-";
        string forecast =
(forecasts != null && idx <
forecasts.Length) ?
Math.Round(forecasts[idx]).ToString() : "-";
        Console.WriteLine($"{userMonth,5}
{actual,10} {forecast,12}");


        SalesForecastOutput salesForecast =
new SalesForecastOutput();
        salesForecast.Actual =
(int)Math.Ceiling((forecasts != null &&
idx < forecasts.Length) ? forecasts[idx] : 0);
        salesForecast.Forecast =
(int)Math.Ceiling((forecasts != null &&
idx < forecasts.Length) ? forecasts[idx] : 0);


        salesForecast.Month = userMonth;
        salesForecasts.Add(salesForecast);
    }


    return Task.FromResult(salesForecasts);
}
```

If you'd like to ensure that the actual and forecasted values closely match, generate sales data with a clear, consistent pattern (such as linear growth, or clear seasonality with limited noise). This ensures that the forecast data generated is almost identical to the actual data trend.

Register the Services in the Program.cs File

The following code snippet shows the Program.cs file. Note how the SalesTimeSeriesForecastService class has been registered with the request processing pipeline.

```
var builder =
WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddScoped
<ISalesTimeSeriesForecastService, ,
SalesTimeSeriesForecastService>();

builder.Services.AddOpenApi();

var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
}
```

```
app.UseAuthorization();


app.MapControllers();


app.Run();
```

Implementing the SalesForecastController Class

The SalesForecast controller provides the API action methods. In this example, there is only one action method. In this class, you create an instance of type ISalesForecastService using dependency injection in the constructor, as shown in the code snippet given below:

```
public class SalesForecastController :
ControllerBase
    {
        private readonly
 ISalesTimeSeriesForecastService
 _salesTimeSeriesForecastService;
        public SalesForecastController
(ISalesTimeSeriesForecastService
 salesTimeSeriesForecast)
        {
            _salesTimeSeriesForecastService =
 salesTimeSeriesForecast;
        }
   }
```

This instance is used in the action method of the SalesForecastController class to retrieve the sales forecasted data of one or more products, as shown below.

```
[HttpGet]
public async Task<List<SalesForecast>>
Get(SalesForecastInput salesForecastInput)
 {
   var salesForecasts =
await _salesTimeSeriesForecastService.
GetSalesForcastData(salesForecastInput);
    return salesForecasts;
}
```

The complete source code of the SalesForecastController class is given in **Listing 10**. That's all you have to do to implement the API. **Figure 12** shows the output looks when you execute the HttpGet action method of this class in Postman.

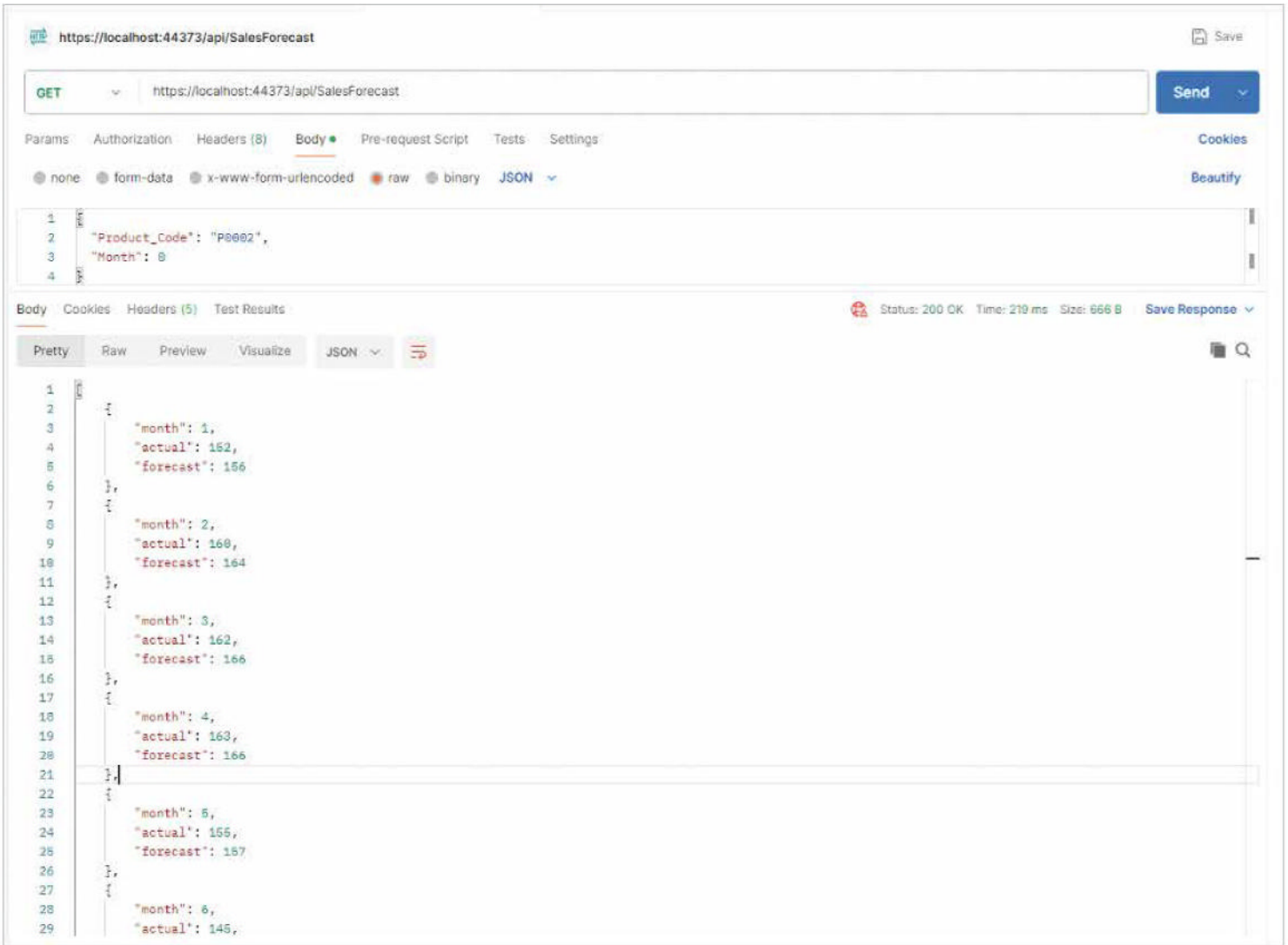


Figure 12: The sales forecast data is generated and shown in Postman.

Listing 10: The SalesForecastController class

```
using Microsoft.AspNetCore.Mvc;
using SalesForecastAPI.Models;

namespace SalesForecastAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class SalesForecastController :
ControllerBase
    {
        private readonly
ISalesTimeSeriesForecastService
_salesTimeSeriesForecastService;
        public SalesForecastController
(ISalesTimeSeriesForecastService
salesTimeSeriesForecast)
        {
            _salesTimeSeriesForecastService =
salesTimeSeriesForecast;
        }


        [HttpGet]
        public async Task<List<SalesForecast>>
Get(SalesForecastInput salesForecastInput)
        {
            var salesForecasts = await
_salesTimeSeriesForecastService.
GetSalesForcastData(salesForecastInput);
            return salesForecasts;
        }
    }
}
```


SPONSORED SIDEBAR:

AI Executive Briefing

Make AI practical for your business. In a focused session with your C-suite, we pinpoint where AI adds value, what to do first, and what it will take. You leave with:

--3–5 prioritized use cases tied to your strategy

--Guidance on data, architecture, security, and skills

--A recommended first pilot and next steps

Learn more or call to schedule:
www.codemag.com/ai

Recommend the Best Product Based on the Generated Sales Forecast Data

Finally, you can modify the API to recommend the best product based on the historical sales data. To achieve this, you should convert the product sales forecasting source code such that the API can recommend the best product to buy based on the historical sales data and ratings. You can add a new column called Rating, which can be the numbers from 1 to 5 with 1 the lowest and 5 the highest. Products that have higher sales usually have higher ratings. The following code snippet shows the change you need in the ProductSalesRecord class.

```
public class ProductSalesRecord
{
    public string Product_Code
    { get; set; }
    public int Year
    { get; set; }
    public int Month
    { get; set; }
    public float Sales
    { get; set; }
    public int Rating
    { get; set; }
}
```

When generating product sales data, the application should include the product ratings (integer values between 1 and 5), correlated with the sales values, i.e., the total number of products sold. Your application should then recommend the best product to buy using a weighted score that's calculated using the sales and rating values of a product. Your application should then apply a data-driven, multi-metric approach to recommend the right product based on the factors mentioned earlier.

Conclusion

With the availability of quantum computing, faster processing, and improved capabilities, machine learning is set to dominate for years (if not decades) to come. You can take advantage of ML.NET to build custom machine learning solutions within the managed environment using C# or F# programming languages and then integrate them into your .NET or .NET Core applications seamlessly. You can get a lot of ML.NET samples here: https://github.com/dotnet/machinelearning-samples

This article helps you understand how to implement prediction and recommendation applications by blending quantitative analytics with historical data, thereby building prediction and recommendation engines that can comprehend customer preferences, improve purchasing efficiency, and help support competitive business strategies in any data-intensive environment.

Joydip Kanjilal

CODE
LEGACY

OLD TECH HOLDING YOU BACK?

Are you being held back by a legacy application that needs to be modernized? We can help. We specialize in converting legacy applications to modern technologies. Whether your application is currently written in Visual Basic, FoxPro, Access, ASP Classic, .NET 1.0, PHP, Delphi… or something else, we can help.

codemag.com/legacy
832-717-4445 ext. 9 • info@codemag.com

©shutterstock

Custom Scripting for Web Applications

In this article, we're going to talk about using a scripting language for developing server-side scripts for web applications. As a scripting language, we'll be using CSCS (Customized Scripting in C#), which is an easy and a lightweight open-source language that has been described in previous CODE Magazine articles: https://www.codemag.com/article/1607081 introduced it, https://www.codemag.com/article/1711081 showed how you can use it on top of Xamarin to create cross-platform native mobile apps, and https://www.codemag.com/article/1903081 showed how you can use it for Unity programming.

Nevio Medancic
nevio@aurasoft.hr

Nevio graduated from University of Ljubljana, Slovenia, in IT and Electrical Engineering. Now he's a Senior Software Developer and Product Designer, with a strong foundation in ERP and Accounting Software. He is CEO at Aura Soft in Croatia.

Enzo Grubisa
enzo@aurasoft.hr

Enzo is a C# developer with experience in WPF, Xamarin. Forms, Xamarin.Android, React, and Node.js. He studied computer science at the Faculty of Engineering at the University of Rijeka, Croatia. He currently works at Aura Soft as a main developer.

Vassili Kaplan
VassiliK@gmail.com

Formerly a Microsoft Lync developer, Vassili has also worked on the Microsoft Maquette Mixed Reality Prototyping Tool and adapted the CSCS scripting language to Microsoft Maquette. He has a Masters in Applied Mathematics with Specialization in Computer Science from Purdue University, in Indiana and a Bachelor in Applied Mathematics from ITAM, Mexico City.

CSCS resembles JavaScript with a few differences, such as the variable and function names being case-insensitive. In this article, we'll be talking about developing web APIs with CSCS. As an abbreviation, we're going to call this version of CSCS as CSCS Web.

Working Principles

CSCS Web is very flexible so it can be used for developing SSR (server-side rendering) or CSR (client-side rendering) applications. An important feature is that you can use existing language functions, or you can also define and create new language functions.

In the current version of CSCS Web, most of the basic functions are implemented. Extensions can be made simply by registering new language functions and then implementing these functions. Later in this article, we're going to show you how to do that.

In SSR applications, you can use templating. In other words, you can predefine HTML pages and use them as templates that will be rendered in a browser. Or you can dynamically format an HTML page that can include HTML code, CSS code, and JavaScript code. In the CSR mode, CSCS Web is mainly used as an API server.

You can also mix the SSR and the CSR code. For example, you can first preload an HTML page that is rendered on the server, and then later, you can proceed with hydrating the whole application on the client side using JavaScript or a JavaScript framework. Below, we will show you how CSCS Web can be used for the server-side rendering of the web applications.

CSCS Web is in a development phase, but any developer can continue using and improving this phase on their own. It's 100% open-source and free to use. Check out the GitHub access links in the sidebar.

Web applications often need APIs with server-side logic for applications to communicate with each other. Servers respond to clients' requests sent to the server's API endpoints.

In this article, such endpoints will be written in CSCS. Interpreted CSCS scripts will use ASP.NET's Minimal API for creating endpoints. CSCS APIs are created as functions, so it's in accordance with the CSCS functional paradigm. When endpoints get called, script functions are executed, and responses are returned. Endpoints, defined in the script, will also be able to accept and access the request's headers and body, work with JSON data, and load, fill, and return HTML templates. We'll also use HTMX as a front-end library that needs a server to generate partial HTML code which, once returned, gets swapped into the DOM.

> Use ASP.NET Core and CSCS together, because sometimes C# just isn't dynamic enough and sometimes you need an extra scripting language to blame.
>
> *Anonymous*

This covers many of the functionalities and gives the opportunity to write APIs inside of the customized scripting language.

Endpoints

In this section, we'll describe how you can create endpoints in CSCS, handle calls to these endpoints, access the request's parameters, and return responses to the client.

> A web endpoint is a specific URL or URI that serves as an entry point for interacting with a web application, service, or API.

Creating Endpoints

For opening an endpoint in CSCS, we implemented the CreateEndpoint() CSCS function. Its usage is like that in the following code snippet:

```
CreateEndpoint("GET", "/",
               "getRootHandlerFunction");
```

You can check how this CSCS function is implemented in **Listing 1**.

Here, the first parameter tells us which HTTP method should be used to invoke this endpoint, the second parameter tells us the route (relative web address) of the endpoint, and the third one tells us the name of the CSCS function that will get called when the endpoint gets called. This CSCS function is defined like this:

```
function getRootHandlerFunction() {
  responseBody = "<html><body><h1>Hello World" +
   " from Web API!</h1><a href='testingPage'>" +
   "Go to testing page</a></body></html>";
  responseHeaders = {«Content-Type": "text/html"};
  statusCode = 200;

  result = { «headers": responseHeaders,
             «body": responseBody,
             "statusCode": statusCode };
  return result;
}
```

This function builds a string containing a simple HTML code that will be returned to the client. When returning the HTML code, the response headers must define the Content-Type to be "text/html" but you can return any type of content. You just need to specify it in the response headers and include it in the response body. For example: application/json, text/plain, image/jpg, etc.

We will go over the details on how to run the sample script later in this article, but you can see in **Figure 1** how the main page looks.

There's a link to the testing page in **Figure 1.** Here's the CSCS function that implements what happens when the user clicks on the "Go to testing page" link (defined in the previous code snippet):

```
CreateEndpoint("GET", "/testingPage",
               "getTestingPageFunction");
function getTestingPageFunction() {
  responseBody = RenderHtml(LoadTemplate(
    ReadConfig("TemplatesDirectory") +
    "testingPage.html"));
  responseHeaders = {«Content-Type»: «text/html»};
  statusCode = 200;

  return Response(responseHeaders,
         responseBody, statusCode);
}
```

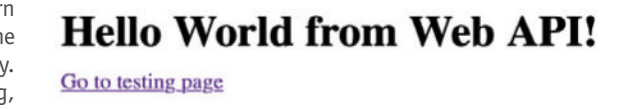


Figure 1: Main screen when opening http://localhost:5058

Listing 1: Implementation of the CreateEndpoint() CSCS Function

```
class CreateEndpointFunction : ParserFunction
{
  private async Task<Variable> ExecScriptFunctionAsync(
        HttpContext context, string scriptFunctionName,
        string httpMethod)
  {
    var requestData = new Variable(Variable.VarType.ARRAY);
    requestData.SetHashVariable("HttpMethod",
          new Variable(context.Request.Method));
    requestData.SetHashVariable("RequestPath",
            new Variable(context.Request.Path));

    var routeParams = new Variable(
           Variable.VarType.ARRAY);
    foreach (var (key, value) in context.Request.RouteValues)
    {
      routeParams.SetHashVariable(key,
             new Variable(value?.ToString()));
    }
    requestData.SetHashVariable("RouteValues", routeParams);

    // Add query parameters
    var queryParams = new Variable(Variable.VarType.ARRAY);
    foreach (var (key, value) in context.Request.Query)
    {
      queryParams.SetHashVariable(key,
           new Variable(value.ToString()));
    }
    requestData.SetHashVariable("QueryParams", queryParams);

    // Add headers
    var headers = new Variable(Variable.VarType.ARRAY);
    foreach (var (key, value) in context.Request.Headers)
    {
      headers.SetHashVariable(key,
         new Variable(value.ToString()));
    }
    requestData.SetHashVariable("Headers", headers);

    // Add body
    var body = Variable.EmptyInstance;
    if (context.Request.ContentLength > 0) {
      try {
        using var reader = new StreamReader(context.Request.Body);
        var bodyContent = await reader.ReadToEndAsync();

        requestData.SetHashVariable("Body",
          new Variable(bodyContent));
      }
      catch(Exception ex) {
        Console.WriteLine(ex.Message);
        requestData.SetHashVariable("Body",
          Variable.EmptyInstance);
      }
    } else {
      requestData.SetHashVariable("Body", Variable.EmptyInstance);
    }

    try {
      return CSCSWebApplication.Interpreter.Run(
       scriptFunctionName, requestData);
    }
    catch (Exception ex) {
      Console.WriteLine(ex.Message);
      return new Variable("Server error.");
    }
  }
}
```

You can also access the endpoint created above directly, by typing **http://localhost:5058/testingPage** in your browser's URL entry field.

Returning Responses

In the previous code snippet, a simple HTML was returned to the client. The responseHeaders dictionary defines headers for the response, the responseBody string contains the body of the response, and the statusCode integer shows the status code being sent to the client.

Another way to create such a dictionary is to use the Response() function. All three of the mentioned variables can be passed to the Response() function that generates a dictionary required by the return statement.

Here's an example:

```
return Response(responseHeaders, responseBody,
                statusCode);
```

Accessing the Request's Headers and Body

Different request's parameters can be accessed in a CSCS script function in the following way:

```
CreateEndpoint("GET", "/testParameters",
               "testParameters");
function testParameters(request) {
  Method = request["HttpMethod"];
  RequestPath = request["RequestPath"];
  Headers = request["Headers"]; // dictionary
  QueryParams = request["QueryParams"]; // dict
  RouteValues = request["RouteValues"]; // dict
  Body = request["Body"]; // string
  //...
}
```

As the function parameter, you can type any variable name and use it later to retrieve the request's arguments. If you need to extract the query parameters, you can do it like in the following CSCS code:

```
CreateEndpoint("GET", "/listQueryParams",
               "listQueryParams");
function listQueryParams(request) {
  QueryParams = request["QueryParams"]; // dict

  for (key : QueryParams.Keys){
    Console.Log(key + " = " + QueryParams[key]);
  }
  //...
}
```

As you can see in the code snippet above, to access any request's query parameter you can use a standard dictionary reading, like QueryParams[key].

But if you need to get the route values, you can access them like in the following CSCS code:

```
CreateEndpoint("GET",
         "/listRouteValues/{value1}/{value2}",
         "listRouteValues");

function listRouteValues(request){
  RouteValues = request["RouteValues"]; // dict
  print("value1=" + RouteValues["value1"]);
  print("value2=" + RouteValues["value2"]);
  //...
}
```

Check out **Listing 1** to see how this CSCS functionality has been implemented.

As you can see, the RouteValues data structure is a dictionary that contains all the values from the request route, and you can access any of them in the usual way.

Working with HTML Templates

In the first example, you saw an endpoint returning a string built inside of the function, but you can also read HTML from a template file. That file can include placeholders that can be filled with actual data, and "if" blocks that get included or excluded from the template depending on a condition.

Loading Templates

To simply load and render a template (a static page, without placeholders or if blocks) you can use the following CSCS code of the testingPage endpoint that was defined above:

```
function getTestingPage() {
  responseBody = RenderHtml(LoadTemplate(
    ReadConfig("TemplatesDirectory") +
               "testingPage.html"));
  ...
}
```

Here, you can also see the ReadConfig() CSCS function that's used to retrieve different config entries from the appsettings.json file via a config entry key.

The LoadTemplate() CSCS function loads the HTML file into memory only and returns an integer representing a handle to this template. This integer is used by other templating functions in this chapter, like the RenderHtml() in the code above.

Filling Template Placeholders

There are also a few placeholder replacement functions that should be called between the LoadTemplate() and RenderHtml() calls. They are the following:

- FillTemplateFromDictionary()
- FillTemplatePlaceholder()
- RenderCondition().

FillTemplateFromDictionary() has as its arguments the template handle mentioned in the previous section and a dictionary with values for placeholders as arguments of the function. It then finds all placeholders that are named as the keys in the supplied dictionary and replaces those placeholders with values from that dictionary. Here's an example of the CSCS script:

```
htmlHndl =
    LoadHtmlTemplate("path_to\\template.html");
valuesDictionary["companyName"] = "My Company";
valuesDictionary["companyAddress"] =
```

```
                "123 Main St, Anytown, USA";

FillTemplateFromDictionary(htmlHndl,
                  valuesDictionary);
```

Let's look at how you can implement any CSCS function and register it with the CSCS interpreter. As an example, let's take a look the implementation of the FillTemplateFromDictionary.

This CSCS function is implemented in C# as follows:

```
class FillTemplateFromDictionaryFunction :
                        ParserFunction {
  protected override Variable Evaluate(
                    ParsingScript script) {
    var args = script.GetFunctionArgs();
    int htmlHndl = Utils.GetSafeInt(args, 0);
    var valuesDict =
        Utils.GetSafeVariable(args, 1);
    var dictKeys = valuesDict.GetKeys();
    var dictVariables = valuesDict.GetAllKeys();

    foreach (var key in dictKeys) {
      var newValue = valuesDict.GetVariable(key);
      Placeholders.ReplaceAll(htmlHndl, key,
                  newValue.AsString());
    }
    return Variable.EmptyInstance;
  }
}
```

The following code registers this function with the CSCS Interpreter in the initialization phase:

```
interpreter.RegisterFunction(
    "FillTemplatePlaceholder",
     new FillTemplatePlaceholderFunction());
```

In the same way, you can define any other function in CSCS. The syntax is the following:

```
interpreter.RegisterFunction("FunctionName",
     new FunctionImplementationClass());
```

Note that the **FunctionImplementationClass** must derive from the **ParserFunction** class.

Take a look at the CSCS repo for details (links in the sidebar).

To add a new CSCS function, follow the following pattern:

1. Implement the functionality in C# by creating a new class, deriving from the ParserFunction class and overriding its Evaluate() method.
2. Register the newly created class with the interpreter like this: interpreter.RegisterFunction("FuncName", new FuncImplementationClass());

And here is an example of an HTML template:

```
<!DOCTYPE html>
<html lang="en">
<body>
    companyName: {{companyName}}
    <br>
    companyAddress: {{companyAddress}}
</body>
</html>
```

Here you can see that the placeholders are defined with the double curly braces.

You can also fill placeholders one at a time with a FillTemplatePlaceholder() CSCS function. Here is an example:

```
FillTemplatePlaceholder(htmlHndl, "companyName",
   "My Company");
FillTemplatePlaceholder(htmlHndl, "companyAddress",
    "123 Main St, Anytown, USA");
```

The FillTemplatePlaceholder() CSCS function is implemented similarly and you're encouraged to take a look at its implementation at https://github.com/AurasoftEnzo/cscs_web/blob/main/CSCSWebFunctions.cs.

Take into the account that CSCS Web is a scripting language so the fewer lines in the script there are (i.e., the more you can do with the templates), the more performant it will be.

Rendering Conditions

There is also a feature to remove or retain code blocks from the template. These code blocks should be surrounded by {{IF_conditionName}}, like in the following example:

```
<!DOCTYPE html>
<html lang="en">
<body>
    {{IF_displayBlock}}
    <p>Lorem ipsum dolor sit amet, consectetur
       adipiscing elit.</p>
    {{IF_displayBlock}}
</body>
</html>
```

Then, by using the RenderCondition() function, this "Lorem Ipsum..." block can be removed or retained. Here is how you can do that:

```
RenderCondition(htmlHndl, "displayBlock", 1 < 2);
```

Here, the first argument is the handle of the template, the second one is the name of the condition from template, and the third one is the actual condition to be evaluated, which will decide whether to retain or remove the code block from the template.

Rendering HTML

Finally, the RenderHtml() function should be called to render the final HTML into a string variable that can then be returned to the client. You can do it as follows:

```
finalHtmlString = RenderHtml(htmlHndl);
```

Here, finalHtmlString holds the whole HTML content.

You can look at the C# implementation of this function at https://github.com/AurasoftEnzo/cscs_web/blob/main/CSCSWebFunctions.cs

Working with JSON

In CSCS Web, there are also a few functions for working with JSON. One function can serialize arrays and dictionaries into JSON strings, another one can deserialize such strings into arrays and dictionaries or even turn the two-dimensional SQL results into JSON strings.

(De)Serializing JSON

When working with CSCS Web, you can turn the CSCS arrays and dictionaries into JSON strings. Suppose your data looks like this:

```
array1 = {};
array1.Add({"key1": "value1", "key2": "value2"});
array1.Add({"key1": "value3", "key2": "value4"});
JSONString = SerializeJson(array1);
```

Then the JSONString will look as follows:

```
[
  {«key1": "value1", "key2": "value2"},
  {«key1": "value3", "key2": "value4"}
]
```

Another way around, if you have a JSONString like this:

```
JSONString = '[{"key1": "value1",
                "key2": "value2"},
               {"key1": "value3",
                "key2": "value4"}]';
```

You can deserialize it into an array or dictionary like this:

```
array2 = DeserializeJson(JSONString);
print(array2[1]["key2"]); // prints value4
```

It probably goes without saying that the CSCS array indices start at 0.

> Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration.
>
> *Stan Kelly-Bootle*

Sql2Json

In CSCS, you can also use SQL connectivity. You probably will often need the SQL results passed back to the client. In the example that follows in the last section, we'll show you some SQL setup, connectivity, and usage.

In all SQL queries, CSCS always returns a two-dimensional array, which has a simple structure: the first row always contains the table column names, and the rest of the rows contain the query data rows. This two-dimensional array will be converted to a JSON string with the use of the Sql2Json() function. This way, you can very easily return data from a database to the client browser. Here is an example from the sample application below which does that:

```
sqlString = "SELECT * FROM employees";
sqlResult = sqlQuery(sqlString);
jsonString = Sql2Json(sqlResult);
```

After executing the above statements, the contents of the jsonString will be the following:

```
[
{"id" : 1,
 "name" : "John",
 "surname" : "Doe",
 "age" : 30,
 "address" : "123 Main St",
 "city" : "Anytown",
 "email" : "john.doe@example.com"
},
{ "id" : 2,
 "name" : "Jane",
 "surname" : "Smith",
 "age" : 25,
 "address" : "456 Elm St",
 "city" : "Othertown",
 "email" : "jane.smith@example.com"
},
…
```

SSR and CSR Paradigm

With the CSCS Web, you can accomplish both the SSR (server-side rendering) and the CSR (client-side rendering). Usually, it isn't enough to prepare just a pure HTML and CSS to have interactive application, so we have two options.

The first one is to enrich the HTML with the hypermedia-driven code. The second one is to use more JavaScript on the client-side. The case that you will use depends on the type of application.

The SSR offers simplicity of coding, especially when there's a lot of interaction with the database. In case you want the SSR application and also want to avoid a lot of JavaScript code, you can use the techniques offered by hypermedia library like HTMX.

For demonstrating the SSR, we'll use HTMX, a lightweight JavaScript library that supercharges the HTML by letting you use the modern web features like AJAX, CSS transitions, WebSockets, and Server-Sent Events directly in your markup using custom attributes.

The front-end will execute the requests from the API and, after it responds with HTMX, it will modify the DOM with received data from the server.

Template + HTMX (SSR)

We have an HTML template file on the server that will be sent to the client. This HTML file looks as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script
    src='https://unpkg.com/htmx.org@1.9.10'>
  </script>
</head>
```

```
<body>
  <div hx-get="/getPartialHtml"
   hx-swap="outerHTML" hx-trigger="load"></div>
</body>
</html>
```

We can return this HTML in a way described in the Load/Render template section. Once the client gets this HTML and the page loads, the HTMX executes a request to /getPartialHtml, which is described as follows:

```
CreateEndpoint("GET", "/getPartialHtml",
               "getPartialHtml");
function getPartialHtml() {
  responseHeaders = {"Content-Type": "text/html"};
  responseBody = "<h1>Hello World!</h1>";

  return Response(responseHeaders,
                  responseBody, 200);
}
```

When the response reaches the client, the HTMX will swap outer HTML of the div element with the received HTML code. We will use this technique in the example section below.

Template + JavaScript (CSR)

With the use of JavaScript, we can make the page client-side rendered. The template file is the following one:

```
<!DOCTYPE
html>
<html lang="en">
<body>
    <script src="/csr.js"></script>
</body>
</html>
```

The corresponding JavaScript file is the following:

```
document.body.innerHTML =
    "<h1>Hello world!</h1>";
```

This makes the JavaScript change the inner HTML of the body when the script is started. It will all be clearer in the sample application that we discuss next.

Sample Application (Employees List)

The sample code for this application can be found in the CSCS Web GitHub repository (see links in the sidebar). This sample entry point is the following: https://github.com/AurasoftEnzo/cscs_web/blob/main/wwwroot/scripts/article2/all.cscs

To be able to start the sample project, you will probably have to change the CSCSConfig section settings in the appsettings.json file. The default version is at https://github.com/AurasoftEnzo/cscs_web/blob/main/appsettings.json

And here is an example of the version to be used if your working environment is macOS:

```
"CSCSConfig": {
  "SQLConnectionString":
  "Data Source=localhost,1433;
   Database=T__DATAX_Y4__BY4;User
   Id=sa;password=Aura2025;
   TrustServerCertificate=True;",

   "ScriptsDirectory":
   "/Users/vass/GitHub/CSCSweb/wwwroot/scripts/",
   "TemplatesDirectory":
   "/Users/vass/GitHub/CSCSweb/wwwroot/templat/",
   "StaticFilesDirectory":
    "/Users/vass/GitHub/CSCSweb/wwwroot/Static/",
   "StartScript": "article2/all.cscs",
}
```

The first thing for this sample to work is to set up SQL Server. You can also do it on a Mac—the easiest is to set it up running inside of a Docker container. Here's one of the possibilities for setting it up: https://devblogs.microsoft.com/azure-sql/development-with-sql-in-containers-on-macos/

Next, let's look at a few snippets.

The body of the HTML template without CSS and JavaScript code is the following:

```
<div class="mb-4 z-50 flex justify-between
 items-center bg-base-100 p-3 rounded-lg
 shadow-lg fixed-header">
 <div class="flex items-center gap-4">
  <h1 class="text-xl font-bold">Employee list</h1>
  <button class='btn btn-accent btn-sm'
   hx-get='/employees/new'
   hx-target='.datagrid-container-master'
   hx-swap='outerHTML'>
        Add New Employee
  </button>
 </div>
</div>
<div class="master-detail-layout">
  <div class="master-table">
   <div class="datagrid-container-master"
    hx-get="/employees" hx-trigger="load"
    hx-swap="innerHTML"></div>
  </div>
</div>
```

The CSCS code for the endpoint that returns a list of the employees is shown in **Listing 2**.

Once you start the application, there are a few different options to test CSCS with Web applications. See **Figure 2** for details, which show the contents of the https://github.com/AurasoftEnzo/cscs_web/blob/main/wwwroot/templates/testingPage.html page.

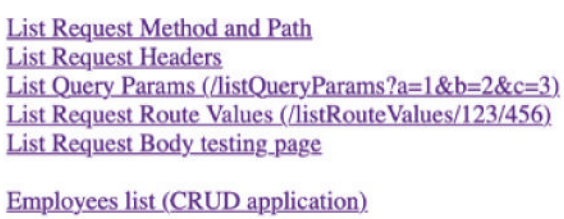

List Request Method and Path
List Request Headers
List Query Params (/listQueryParams?a=1&b=2&c=3)
List Request Route Values (/listRouteValues/123/456)
List Request Body testing page
Employees list (CRUD application)

Figure 2: Different options to test CSCS WEB

Listing 2: Sample Code of the getEmployees() CSCS function

```
CreateEndpoint("GET", "/employees", "getEmployees");
function getEmployees(request) {
  // Pagination
  employees_page = 1;
  if (Contains(request["QueryParams"], "page")) {
    employees_page = int(request["QueryParams"]["page"]);
  }
  if (employees_page < 1) {
    getEmployees_responseHeaders = {"Content-Type": "text/html"};
    return Response(getEmployees_responseHeaders,
      "Error: 'page' must be positive integer", 200);
  }

  // Sorting
  employees_sort = "id";
  if (Contains(request["QueryParams"], "sort")) {
    employees_sort = request["QueryParams"]["sort"];
  }

  employees_order = "asc";
  if (Contains(request["QueryParams"], "order")) {
    employees_order = request["QueryParams"]["order"];
  }

  // Fixed page size
  employees_pageSize = 10;
  employees_skip = (employees_page - 1) * employees_pageSize;

  //alert after html
  employees_alert = "";
  if (Contains(request["QueryParams"], "alertText")) {
    employees_alert = request["QueryParams"]["alertText"];
  }
  employees_query = "SELECT " +
     "id, name, surname, age, address, city, email " +
     "FROM employees " +
     "ORDER BY " + employees_sort + " " + employees_order+" "+
     "OFFSET @skip ROWS FETCH NEXT @pageSize ROWS ONLY";
  employees_sqlParams = {};
  employees_sqlParams.Add({"@skip", employees_skip});
  employees_sqlParams.Add({"@pageSize", employees_pageSize});
  employees_records = sqlQuery(employees_query,
                      employees_sqlParams);

  employees_countQuery = "SELECT COUNT(*) FROM employees";
  employees_countResult = sqlQuery(employees_countQuery);
  employees_totalRecords = employees_countResult[1][0];
  employees_totalPages = Math.Ceil(employees_totalRecords /
                         employees_pageSize);

  // Build HTML
  employees_html = "<div class='datagrid-container-master'>";
  employees_html += "<table class='datagrid-table'>";
  employees_html += "<thead>";
  employees_html += "<tr>";
  employees_newOrder = "asc";
  if (employees_sort == "id" && employees_order == "asc") {
    employees_newOrder = "desc";
  }

  employees_html += "<th><a class='link' hx-get=
   '/employees?page=1&sort=id&order=" + employees_newOrder +
   "' hx-target='.datagrid-container-master'
   hx-swap='outerHTML'>Id</a></th>";

  for(employees_i = 1; employees_i < Size(employees_columns);
      employees_i++) {
    employees_column = employees_columns[employees_i];

    employees_newOrder = "asc";
    if (employees_sort == employees_column &&
        employees_order == "asc") {
      employees_newOrder = "desc";
    }

    employees_html += "<th><a class='link'
      hx-get='/employees?page=1&sort=" + employees_column +
      "&order=" + employees_newOrder +
      "' hx-target='.datagrid-container-master'
       hx-swap='outerHTML'>" + employees_column +
       "</a></th>";
  }

  employees_column = "Actions";
  employees_html += "<th>" + employees_column + "</th>";
  employees_html += "</tr>";
  employees_html += "</thead>";
  employees_html += "<tbody>";

  if(employees_records != null && Size(employees_records) > 1) {
    for(employees_i = 1; employees_i < Size(employees_records);
        employees_i++) {
      employees_row = employees_records[employees_i];
      if (employees_i % 2 == 0) {
        employees_html += "<tr data-id='" + employees_row[0] +
      "' class='bg-base-200 hover:bg-base-300 cursor-pointer' >";
      } else {
        employees_html += "<tr data-id='" + employees_row[0] +
        "' class='hover:bg-base-300 cursor-pointer' >";
      }

      employees_html += "<td data-field='id'>" + employees_row[0]
        + "</td>";
      employees_html += "<td data-field='name'
       class='text-center'>" + employees_row[1] + "</td>";
      employees_html += "<td data-field='surname'
        class='text-center'>" + employees_row[2] + "</td>";
      employees_html += "<td data-field='age'
        class='text-left'>" + employees_row[3] + "</td>";
      employees_html += "<td data-field='address'
        class='text-center'>" + employees_row[4] + "</td>";
      employees_html += "<td data-field='city'
        class='text-center'>" + employees_row[5] + "</td>";
      employees_html += "<td data-field='email'
        class='text-center'>" + employees_row[6] + "</td>";
      employees_html += "<td class='flex gap-1'>";
      employees_html += "<button class='btn btn-info btn-xs
        action-button' hx-get='/employees/" + employees_row[0] +
        "/edit' hx-target='.datagrid-container-master'
         hx-swap='outerHTML'>Edit</button>";
      employees_html += "<button class='btn btn-error btn-xs
        action-button' onclick='confirmDeleteEmployee(" +
        employees_row[0] + ")'>Delete</button>";
      employees_html += "</td>";
      employees_html += "</tr>";
    }
  }
  employees_html += "</tbody>";
  employees_html += "</table>";

  // Pagination
  employees_html += "<div class='pagination'>";
  if(employees_page > 1) {
    employees_html += "<a class='btn btn-sm' hx-
     get='/employees?page=" + (employees_page - 1) + "&sort=" +
     employees_sort + "&order=" + employees_order +
       "' hx-target='.datagrid-container-master' hx-
      swap='outerHTML'>Previous</a>";
  }
  employees_html += "<span class='page-info'>Page " +
```

Listing 2: continued...

```
    employees_page + " of " + employees_totalPages + "</span>";
  if(employees_page < employees_totalPages) {
    employees_html += "<a class='btn btn-sm' hx-
    get='/employees?page=" + (employees_page + 1) + "&sort=" +
    employees_sort + "&order=" + employees_order +
      "' hx-target='.datagrid-container-master'
      hx-swap='outerHTML'>Next</a>";
  }
  employees_html += "</div>";
  employees_html += "</div>";
  employees_html += employees_alert;

  getEmployees_responseHeaders_2 = {"Content-Type":"text/html"};
  return Response(getEmployees_responseHeaders_2,
                  employees_html, 200);
}
```

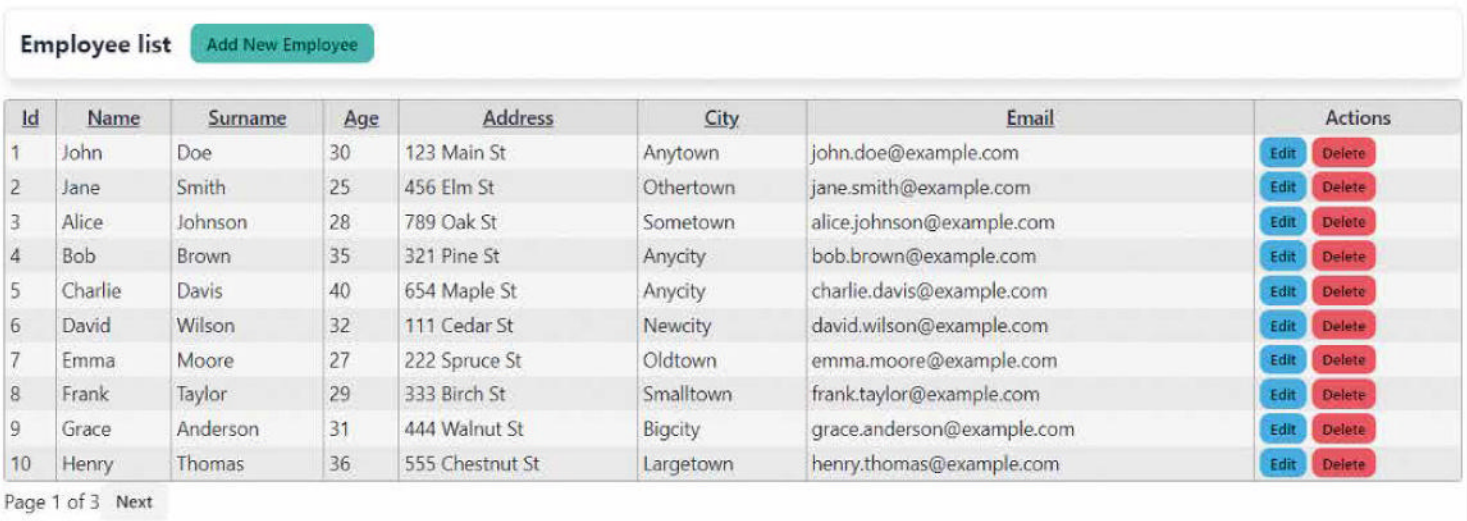

Employee list
Add New Employee
Id Name Surname Age Address City Email Actions
1 John Doe 30 123 Main St Anytown john.doe@example.com Edit Delete
2 Jane Smith 25 456 Elm St Othertown jane.smith@example.com Edit Delete
3 Alice Johnson 28 789 Oak St Sometown alice.johnson@example.com Edit Delete
4 Bob Brown 35 321 Pine St Anycity bob.brown@example.com Edit Delete
5 Charlie Davis 40 654 Maple St Anycity charlie.davis@example.com Edit Delete
6 David Wilson 32 111 Cedar St Newcity david.wilson@example.com Edit Delete
7 Emma Moore 27 222 Spruce St Oldtown emma.moore@example.com Edit Delete
8 Frank Taylor 29 333 Birch St Smalltown frank.taylor@example.com Edit Delete
9 Grace Anderson 31 444 Walnut St Bigcity grace.anderson@example.com Edit Delete
10 Henry Thomas 36 555 Chestnut St Largetown henry.thomas@example.com Edit Delete
Page 1 of 3 Next

Figure 3: The browser view of the list of employees

The main example, and the subject of this article, is the last one, the Employee List. Once you click on it, you will get a view as in **Figure 3**.

If you click on the "Add New Employee" button, you'll get a new page, as in **Figure 4**. You will also get a similar view if you click on the Edit button for any user.

Once you fill out all the details and click on the Save button, you'll get the contents shown in **Figure 5**. You'll also get a similar view if you click on the Edit button for any user.

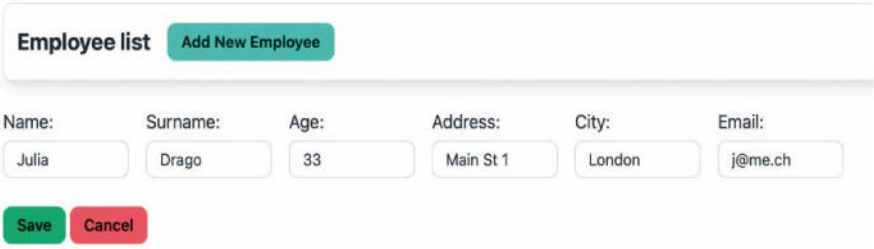


Figure 4: Adding a new user

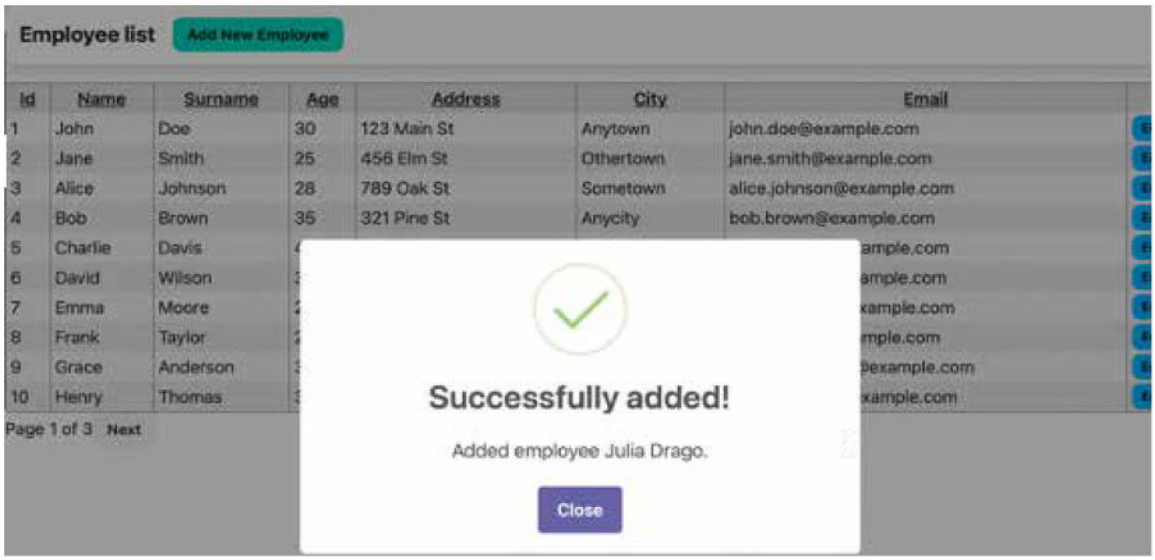


Figure 5: A pop-up message when a new user was added

Listing 3: Sample Code of the createEmployee() CSCS function

```
CreateEndpoint("POST", "/employees", "createEmployee");
function createEmployee(args) {
  create_fields = [
    "Name", "Surname", "Age", "Address", "City", "Email"
  ];

  create_values = {};
  for(create_i = 0; create_i < Size(create_fields); create_i++) {
    create_field = create_fields[create_i];
    create_values[create_i]= GetValueFromForm(args["Body"],
       create_field);
  }

  create_values2 = {};
  create_values2["Name"] = GetValueFromForm(args["Body"], "Name");
  create_values2["Surname"] = GetValueFromForm(args["Body"],
          "Surname");
  create_values2["Age"] = GetValueFromForm(args["Body"], "Age");
  create_values2["Address"] = GetValueFromForm(args["Body"],
          "Address");
  create_values2["City"] = GetValueFromForm(args["Body"], "City");
  create_values2["Email"]= GetValueFromForm(args["Body"],"Email");

  if (!IsInt(create_values2["age"])) {
    create_headers = {"Content-Type": "text/html"};
    create_employees_alert = "<script>Swal.fire({title:
      'ERROR!', text: 'Age must be an integer.', icon: 'error',
       confirmButtonText: 'Close'});</script>";
    create_employees_validationError_html =
      getNewEmployeeFormWithValues(create_values2,
      create_employees_alert);

    return Response(create_headers,
        create_employees_validationError_html, 200);
  }

  create_query = "INSERT INTO employees (";
  create_first = true;
  for(create_i = 0; create_i < Size(create_fields); create_i++) {
    if (!create_first){
      create_query += ", ";
    }
    create_query += create_fields[create_i];
    create_first = false;
  }

  create_query += ") VALUES (";
  create_first = true;
  for(create_i = 0; create_i < Size(create_fields); create_i++) {
    if (!create_first) {
      create_query += ", ";
    }

    create_query += "@" + create_fields[create_i];
        create_first = false;
  }
  create_query += ")";
  create_sqlParams = {};
  for(create_i = 0; create_i < Size(create_fields); create_i++) {
    create_sqlParams.Add({"@"+create_fields[create_i],
          create_values[create_i]});
  }
  sqlNonQuery(create_query, create_sqlParams);

  create_args2 = {};
  create_args2["QueryParams"] = {"page": 1};

  create_employees_alert = "<script>Swal.fire({title:
    'Successfully added!', text: 'Added employee " +
     create_values2["name"] +  " "+ create_values2["surname"] +
   ".', icon: 'success', confirmButtonText: 'Close'});</script>";

  create_args2["QueryParams"] = {"alertText":
     create_employees_alert};

  return getEmployees(create_args2);
}
```

Pretty cool, right? But the coolest part of this is that everything has been created using the CSCS Web scripting language. Look at **Listing 2** on how to get a list of all employees in CSCS and at **Listing 3** to check out how adding a new user has been implemented in CSCS.

Wrapping Up

In this article, you saw how to use the CSCS scripting language in ASP.NET Core. The main advantage is that you can add any functionality to your project without the need for recompilation, because all the CSCS scripts are loaded at runtime. Of course, it could negatively affect your performance, so you should make sure that there are no complex mathematical computations done in CSCS, but rather mostly some GUI-related code.

> ASP.NET Core with CSCS? That's a Swiss Army knife—if the knife occasionally compiles itself.
>
> *Anonymous*

In this article, we've been talking mostly about just one example, a list of employees. There are many more interesting examples, see **Figure 2** for the full list.

All of these examples are implemented in the https://github.com/AurasoftEnzo/cscs_web/blob/main/www-root/scripts/article2/all.cscs file.

We're looking forward to your feedback: specifically, your experience in applications where you're using CSCS Web, and what other features in CSCS you would like to see.

Nevio Medancic, Enzo Grubisa, Vassili Kaplan

CODE

References

CSCS Repository: https://github.com/vassilych/cscs

CSCS Web Repository: https://github.com/AurasoftEnzo/cscs_web

HTMX: https://htmx.org/

CSCS Visual Studio Code Debugger Extension: https://marketplace.visualstudio.com/items?itemName=vassilik.cscs-debugger

MCP: Building the Bridge Between AI and the Real World

AI is like working with an inept wizard. (Yes, I have a lot of metaphors for this.) When you ask the wizard a question, he responds with the intellect and rapidity of someone who has access to the knowledge of the cosmos. He's read everything, but he's a bit dotty. He's lived his entire life in his lair, consuming his tomes. Despite his vast knowledge, he has no idea what happened in the

world yesterday. He doesn't know what's in your inbox. Moreover, he knows nothing about your contact list, your company's proprietary data, or the fact that your cousin's birthday party got bumped to next Friday. The wizard is a genius. He's also an idiot savant.

Therein lies the paradox. We have designed amazing tools, but they require a lot of handholding. Context has to be spoon-fed. You can paste an entire mountain of reference documents and a virtual novel of a prompt. That amount of work can often eliminate any benefit you get from using an LLM at all. When it does work, it's a victory but it feels like you've wrestled the LLM into submission instead of working with it.

Users have been cobbling together ad hoc solutions for this problem. Plug-ins. Vector databases. Retrieval systems. These Band-Aids are clever, but fragile. They don't cooperate with each other. They break when you switch providers. It's less "responsible plumbing" and more "duct tape and prayer."

This is where Model Context Protocol (MCP) comes in. It establishes a foundational infrastructure rather than creating one more marketplace for custom connectors. MCP sets up standardized rails for integrating context. This shared framework enables models to request context, retrieve it from authorized sources, and securely use it. It replaces the current kluge of vendor-specific solutions with a unified protocol designed to connect AI to real world systems and data.

As AI transitions from an experimental novelty to practical infrastructure, this utility becomes crucial. For the wizard to be effective, he needs to be able to do more than solve one-off code hiccups or create content for your blog. For true usefulness at scale in a professional environment you need a standardized way to integrate context. That context has to respect permissions, meet security standards, and be up to date.

The Problem of Context in AI

Models tend to make things up and they do it with confidence. Sometimes they cite fictional academic papers. Sometimes they invent dates, statistics, or even people. These hallucinations are a huge problem, of course, but they're a symptom of a much larger issue: a lack of context.

The Context Window Problem

Developers have been trying to develop workarounds by providing relevant data as needed. Pasting in documents, providing chunks of a database, and formulating absurdly robust prompts. These fixes are great, but every LLM has what we call a context window. The window determines how many tokens a model can remember at any given time. Some of the bigger LLMs have windows that can accommodate hundreds of thousands of tokens, but users still quickly find ways to hit that wall.

Bigger context windows should be the answer, right? But there's our Catch 22: The more data you provide within that window, the more fragile the entire set up becomes. If there's not enough context, the model may very well just make stuff up. If you provide too much, the model bogs down or becomes too pricey to run.

The Patchwork Fixes

The AI community wasn't content to wait for one of the big players to provide a solution. Everyone rushed to be first-to-market with an assortment of potential fixes.

Custom plug-ins let the models access external tools and databases, extending their abilities beyond the frozen training data. You can see the issue here. Plug-ins designed for one platform won't work with another. Your workspace becomes siloed and fragmented, forcing you to rework your integrations if you try to switch AI providers.

Retrieval Augmented Generation (RAG) converts documents to embed them into a vector database so that you can pull only the most relevant chunks during a query. This method is pretty effective but requires significant technical skills and ongoing fine-tuning based on your organization's specific requirements.

Custom APIs and middleware are bespoke methods that offer maximum control so that the model can converse with your systems, but the application of these is targeted. These fixes can't be easily adapted or migrated.

Each of these approaches are useful, but they all suffer from the same weakness as any proprietary solution. The developers have to reinvent the wheel each time. Without universal integration standards, these solutions are unstable and non-transferable. AI systems need a standardized approach for context access and authentication.

Why Context Is Absolutely Necessary

This is about more than convenience and developer hours. AI systems context draws a line between novelty and actual functional infrastructure.

Jason Murphy
Thestrangerous.substack.com
@jason-murphy.bsky.social

Jason is a writer, AI enthusiast, and media creator. As the producer and co-host of **Hacking the System** on the National Geographic Channel, he stole cars in Hollywood, made improvised smoke bombs, and prepared for the apocalypse. On YouTube, he co-created and hosted **the Modern Rogue**, where he explored hacking, lock-picking, and tradecraft. After publishing multiple speculative fiction novels and writing a produced screenplay, Jason is currently exiled in the desert, where he stares into the future with awe and terror. And he still wants to make a video game.

- **Business:** In a business setting, context enables an AI assistant to provide an answer shaped by the customer's actual account data. Otherwise, the model could return a general response or even a hallucination.
- **Healthcare:** Anchoring responses within the context of actual patient data and medical records is the only responsible reply. Context is the difference between safe clinical conclusions and dangerous guesswork.
- **Education:** In education, a one-size-fits-all approach is far less effective than a custom tutor. Context provides an AI tutor with the guidelines to teach from specific materials based on the student's needs.

Context sharpens results and reduces responses aimed squarely at the middle. It turns your wizard-in-a-box into a reliable collaborator familiar with your workflow.

Why a Protocol Is Needed

A RAG here. A plug-in there. The fixes multiply with every change or addition to your workflow, and when you're using AI as the tech expands and adapts, the changes come fast and frequently. Before long, your workflow becomes brittle and untenable.

The duct tape that binds this all together doesn't scale. It's an inelegant and temporary fix for a large, complicated solution. As the industry grapples with finding the true utility of this nascent technology, the need for a protocol becomes obvious. It requires a shared and open standard that determines:

- How those models ask for data
- How those services respond
- How permissions are managed

Without common communication standards, we couldn't even use the internet. HTTP and TCP/IP prevent us from having to use a different browser for every website. That's the state of AI models now. It's a tangle of proprietary integrations, bespoke connections, and workarounds. Standardized protocols will turn it into something capable and reliable.

> The state of AI models now is a tangle of proprietary integrations, bespoke connections, and workarounds.

What Is Model Context Protocol?

Model Context Protocol (MCP) is one of the first systematic approaches to a solution for this patchwork.

MCP functions as an open standard. It's like a universal translator between AI models and your systems. It defines how the model discovers tools, requests information, manages access controls, and uses context. This shared approach replaces the mess of case-by-case fixes with broad compatibility.

The Origins of MCP

MCP was born from years of trial-and-error with all the messy plug-ins, APIs, and middleware. Developers at Anthropic saw the same recurring issues. The idea was clear and uncomplicated: design a protocol—not just a library or an SDK, but an open standard accessible to anyone. It should be able to be implemented across various vendors, models, and tools. MCP is less like "another product from Anthropic" and more like an attempt to establish a shared ecosystem.

The Core Principles of MCP

MCP was designed with a few guiding principles at its base: interoperability, security and permissions, transparency, and flexibility.

One of MCP's primary strengths is its commitment to seamless interoperability across the entire AI ecosystem. It's not tied to any one AI vendor. Any model that supports MCP can request context from any MCP-compliant source. That could be a proprietary in-house tool, but it could also be something like a Salesforce database or a Google calendar.

In MCP's architecture, security and user control are paramount. Of course, allowing an AI to query external systems without guardrails is a recipe for disaster. A very big disaster that happens very quickly. It could trigger too many API calls, violate compliance regulations, and on and on. MCP, however, builds in a permissions model. The administrator approves what sources the AI has available to it, what information can be accessed, and what specific actions it can take.

Knowing those sources is critical. When models pull from data sources without the users understanding the origins of that data, it creates significant vulnerabilities. MCP puts a need for transparency at the forefront by focusing on full visibility. Users can see which sources are being accessed for each call and can track how that contextual data was used in the responses.

The protocol isn't bound to just streaming data. With flexibility in mind, it was built to work with archived documents, databases, PDFs, and so on. With this broad architecture, it's able to adapt and evolve as new use cases present themselves.

A Working Definition

Model Context Protocol is an open standard for safely and visibly connecting AI models to external context sources—including data, tools, and actions—regardless of the underlying technology, without locking into a single vendor.

Simply put, MCP is a unified connector that builds bridges between AI models and your digital infrastructure.

How MCP Works

If we set the abstractions aside, we can take a look at how it actually works in practice. When a model requests context, what happens under the hood? How does MCP manage the process with the key principles in mind?

First, we break it down step-by-step.

Glossary of MCP Terms

Client: The MCP component embedded in your AI application that acts as a translator and gatekeeper. It translates the AI model's requests into the standard MCP format while handling security, permissions, and connections to external data sources.

Server: The data source or tool on the other end of an MCP connection. Each server tells clients what it can do, then fulfills requests for information or actions. Examples range from calendar and database servers to specialized business tools.

Capability Declaration: What a server announces it can do. Like a restaurant menu, it lists available services with clear yes/no permissions (read_availability: true, create_event: true). This prevents confusion and blocks unauthorized requests.

Context Window: How much information a model can hold in its working memory at one time. Bigger windows mean more data, but MCP offers a better solution: smart, secure access to exactly the context you need, when you need it, without hitting limits.

A High-Level Look

The process of using MCP is a handshake in four parts.

First, the model asks for context. This isn't a blind request. The AI generates a structured and specific request, following MCP's format: "I need information about X" or "I need to trigger this operation to do Y." You can see the basic JSON for a client request format in this snippet:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/call",
  "params": {
    "name": "check_availability",
    "arguments": {
      "date": "2024-01-10",
      "time_range": "morning"
    }

  }

}
```

The MCP client then handles the request. This client rests inside the application running the model—a chat interface, a workflow engine, etc. Rather than immediately and blindly passing things through, it analyzes the request. Sources are checked. The client manages the list of available servers. It handles authentications and permissions and translates between the model's requests and the standardized MCP calls. Take a look at the next listing to see an example of server capability declaration.

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "protocolVersion": "2024-11-05",
    "capabilities": {
      "resources": {
        "subscribe": true,
        "listChanged": true
      },
      "tools": {
        "listChanged": true
      },
      "prompts": {
        "listChanged": true
      }
    },
    "serverInfo": {
      "name": "calendar-server",
      "version": "1.0.0",
      "description": "A server for managing
 calendar events and availability"
    }
  }
}
```

After the client works its magic, the MCP servers provide the context. For our purposes, the server is any external system that has implemented MCP. These systems implement the MCP specifications, like announcing its capabilities, accepting structured requests, or returning structured responses. This could range from the simple to the complex, like a database server exposing product inventory, a calendar server with schedule availability, or a tool server that does things like sending emails or booking appointments. After the client reaches out, information is returned using MCP's standardized response format. The Server Response Structure can be seen here:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "Available slots for
Wednesday morning:\n- 9:00 AM - 10:00 A
M\n- 10:30 AM - 11:30 AM\n- 11:45 AM - 12:00 PM"
      }
    ],
    "isError": false
  }
}
```

Finally, the model then incorporates the context into its reasoning. It doesn't just get a chunk of anonymous raw text. The MCP defines the information provided with structure and formatting. This could be JSON objects for structured data, metadata to tell the user where the data was sourced from, or permissions labels for what the mod-

Listing 1: Server Capability Declaration

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "protocolVersion": "2024-11-05",
    "capabilities": {
      "resources": {
        "subscribe": true,
        "listChanged": true
      },
      "tools": {
        "listChanged": true
      },
      "prompts": {
        "listChanged": false
      }
    },
    "serverInfo": {
      "name": "calendar-server",
      "version": "1.0.0",
      "description": "A server for
 managing calendar events and availability",
      "permissions": {
      "allowedOperations":
["read", "create"],
        "restrictedOperations":
["delete", "modify_all"],
        "dataScope": "user_calendar_only",
        "requiresConfirmation":
["create", "modify"]
      }
    }
  }
}
```

el can or cannot see. This makes the context structured, traceable, and composable.

That's the loop. Model > Client > Server > Model.

Example Workflow

Let's walk through a hypothetical:

Scenario: You ask your AI assistant, "Can you reschedule my meeting with Dr. Doom for Wednesday morning?"

Here's what happens with MCP:

1. The model evaluates the request. It identifies that calendar access is required.
2. The client checks permissions. Is a calendar MCP server connected? Does the user have permission for this action? **Listing 1** shows a partial example of how server capabilities are declared with permissions.
3. The client relays the request. It asks the calendar server: "What slots are open for Wednesday morning?"
4. The server sends the results. It returns structured availability information like time slots, conflicts, and any included metadata.
5. The model analyzes the context. It selects the most appropriate time slot and builds the response.
6. The client triggers an action. If the user confirms, the client signals the server to update the calendar.

All these steps implement MCP's core principles, as referenced earlier. With MCPs security protocols, the model is unable to ransack your data. Each and every request first goes through the client and can only take action if the server indicates that it's allowed do so.

The ability to audit the sources removes any mystery. No longer are you at the whim of black-box AI references.

Let's say you want to stop using Google Calendar and switch over to Outlook. This change won't require deep tinkering of your custom application. As long as both use MCP servers, the model can take advantage of the tenet of interoperability.

In terms of scalability, it's easy to supplement your new context sources without rebuilding the entire integration. That could include features like booking systems, availability checkers, or multiple calendar platforms, all without extensive alterations to your architecture.

It's Not Just Data; It's Actions and Tools

The context I keep referring to isn't just a list of rules or static information. This is inherent to MCP's design philosophy. An MCP server doesn't just return data—it can perform operations. This enables your LLM model to both analyze your calendar and create appointments.

For example:

- A project management server might not just return task lists, but also delegate assignments and reschedule due dates.
- A social media server might not just provide engagement metrics but also post new content and respond to your online community.

Security, Privacy, and Governance

Is MCP the holy grail of making functional AI agents—without using a bottle of Elmer's glue and some yarn to bundle the integrations together? If your internal security alarms are ringing, that just means you're thinking responsibly. Every time I read about one of these new AI applications, I cringe at the security implications. A workflow that makes it easier to move context across systems also expands your exposure surface. Such a system has to prioritize security, privacy, and governance.

Security by Design

What sets Model Context Protocol apart from the previous hacks is that security is not an afterthought. It's built into the protocol from the ground up.

With a focus on permissioned access, the model can't just do whatever it thinks best. Every context request goes through the MCP client. That client makes sure the rules are followed, specifically about what sources are used and what actions the model can take.

The capabilities of MCP servers are scoped and must be declared up front. If a server authorizes only read-only access to specific records listed, then it can only operate within those bounds. It can't escalate into catastrophic decisions.

Both the MCP servers and the clients handle authentication with each other before any data can be exchanged. This way spoofed or rogue servers are unable to return tainted or poisoned context.

Practically speaking, the AI assistant can't make a wild choice dance around in your payroll system or sensitive HR documents. Only explicitly connected and approved sources can be used.

Privacy and Transparency

The second pillar to consider is privacy. There's a very real fear that AI assistants can rapidly erode privacy protections and put sensitive data into the wrong hands. MCP aims to address that with a focus on transparency.

As mentioned previously, each piece of context that the model gets also comes with metadata regarding its origin. The answer comes with sources, like any good research paper.

End users and administrators are also given complete control. They can decide which servers are connected and what specific data can be accessed. If a user doesn't need or want their email accessible, the model is locked out of that domain.

If necessary, all the requests and responses can be logged. No mysterious decisions made without oversight. This establishes a trail that users can audit. What context was accessed and when is not a mystery. In sectors like healthcare and finance, which are laden with compliance regulations, this is critical.

The Governance Layer

Although security and privacy are vital, use of MCP brings up some complicated questions regarding governance. We're still in the Wild West phase of AI and as it

Quick Tips

Keep It Simple

Build a rudimentary MCP server that announces basic capabilities—for example, a **calendar server** that exposes read_availability and create_event.

Focus on Permissions

Keep your scope narrowly defined. Grant read-only access to a small dataset (like one calendar or one project board). This helps you understand MCP's guardrails without juggling too many things at once.

Expand Bit by Bit

Once the handshake loop (Model > Client > Server > Model) works, you can expand your capabilities by adding more context sources: databases, file systems, or APIs.

continues to evolve, we'll remain there. It can be a crap shoot determining which servers can be trusted. How can an organization of any size know where to set boundaries? How do we determine what the model is allowed to access?

From a high level, there are three different layers to governance. The technical side has already been amply covered above. MCP enforces the structure, permissions, and the authentication. But what about organizational governance? Companies and institutions have to push forward themselves and determine what servers to connect to, along with what data to expose and how to audit the use of that data.

Technology outpaces governance. It always will. As MCP-enabled AI gets its proverbial hands dirty in very sensitive sectors like healthcare and banking, regulators and legislators will step in. Right now, MCP is in its infancy, so the lion's share of governance is happening at the organizational layer. Hard questions will need to be asked. What's acceptable context? How is consent managed?

Risks and Threat Models

Of course, there are not and likely never will be any protocols that are completely risk-free. Although Model Context Protocol reduces some of those threats, there are certainly plenty of dangers that remain.

- **Context Poisoning:** If a malicious actor can compromise an MCP server, they can manipulate the data flowing to the model, corrupting it. Transparency can provide visibility to the data, but it's unable to filter out tainted information.
- **Overreach:** It's tempting for an organization to default to maximum connectivity. Maybe it opts to give the AI assistant far more access than is truly needed. That plants the seeds for an inevitable breakdown in governance.
- **Surveillance Misuse:** The protocol has no inherent bias, but the use of it will define the outcomes. There's always a chance of abuse. In such a scenario, a malicious user could weaponize MCP to aggregate and surveil sensitive user information.
- **Ecosystem Fragmentation:** There's always the possibility that MCP won't be fully adopted but cloned. MCP-like variations could fragment the landscape and cut compliance corners. Interoperability breaks down and erodes security assurances.

Juggling Openness and Safety

Therein lies the friction: The openness and flexibility of MCP leads to a more powerful ecosystem. But with that openness comes increased risk. How are servers vetted? Soon, we'll see them popping up all over the place. Some of them will be compromised. It's just the law of numbers. How can users ensure that these upstart servers won't leak, corrupt, or abuse data?

We've been down this road before. Just as HTTP made the web possible, it also introduced us to all the wonderful things we keep our eyes open for every day, like phishing schemes, spam, and malware, for starters. Protocols such as MCP cannot guarantee safety. It's the job of governance to steer the environment toward productive, responsible use.

None of these criticisms should be a dealbreaker. They simply serve as a reminder that MCP is just a protocol, not a cure-all. Although it does solve the very real problem of context fragmentation, it also introduces its own set of challenges.

As with any nascent and incredibly powerful technology, the healthiest approach to take is one of reserved confidence. MCP gives us a way to construct more reliable and transparent artificial intelligence systems, but it certainly can't guarantee safety or proper governance. Like any plumbing, it only works when built upon a solid foundation, managed by human hands.

Jason Murphy
CODE

SPONSORED SIDEBAR:

Adding LLM Assistants to Your Apps

The future is here now and you don't want to get left behind. Unlock the **true potential** of your software applications by adding **AI Assistants**.

CODE Consulting can assess your applications and provide you with a roadmap for adding LLM features and optionally assist you in adding them to your applications.

Reach out to us today to get your application assessment scheduled.
www.codemag.com/ai

Fixing Security Issues in Seconds, Not Sprints, Powered by GenAI

In modern software development, speed is everything. Teams face constant pressure to deliver features faster while maintaining high quality and security standards. Unfortunately, security often lags behind. Developers typically discover vulnerabilities during pull request reviews, or using tools like CodeQL or SonarQube to flag them later—at a point when fixing them takes more time and costs more. What if developers could catch those issues instantly, right in their local environment, before even submitting a PR? That's the idea behind **SecureCodeAgent**, a GenAI-powered solution that shifts security left in the software development lifecycle.

Gaurav Mittal
Gauravmittal1985@gmail.com

Gaurav Mittal is an accomplished author and international speaker, recognized for his published articles. He has spoken at global conferences and served on several judging panels. In addition to his professional achievements, Gaurav actively contributes to non-profit organizations through volunteer work. Outside of work, he enjoys spending time with his children and playing sports.

Generative AI (GenAI) is the broad technology category. OpenAI builds some of the leading GenAI models, and Microsoft's Azure OpenAI API provides enterprise access to those models. SecureCodeAgent uses this service as its GenAI engine. In this article, I use the Azure OpenAI API—Microsoft's implementation of OpenAI's models—as an example of GenAI in practice. This allows developers to run real-time code scans as they write. It identifies insecure patterns, highlights vulnerable lines, suggests safer alternatives, and even assigns severity levels. This way, developers receive immediate, actionable feedback without waiting for formal reviews or external tools.

In this article, I'll demonstrate how SecureCodeAgent works, show examples of secure coding in action, and walk through generating a workflow that automates these checks. See **Figure 1** for quick summary of secure code agent. By the end, you'll see how this approach strengthens security while keeping development fast and efficient.

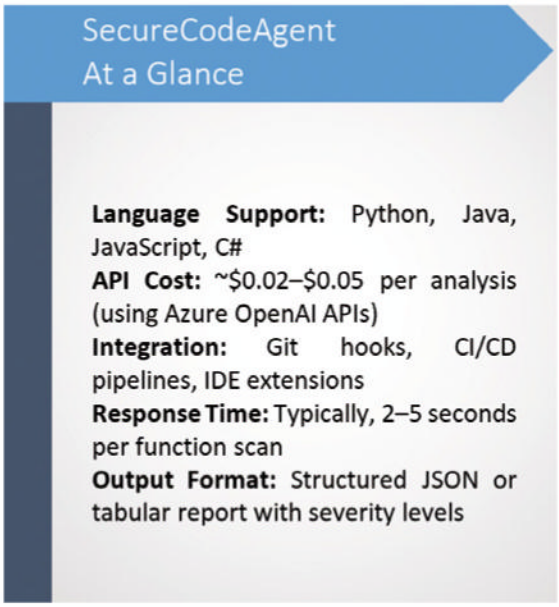

SecureCodeAgent
At a Glance
Language Support: Python, Java, JavaScript, C#
API Cost: ~$0.02–$0.05 per analysis (using Azure OpenAI APIs)
Integration: Git hooks, CI/CD pipelines, IDE extensions
Response Time: Typically, 2–5 seconds per function scan
Output Format: Structured JSON or tabular report with severity levels

Figure 1: SecureCodeAgent at a glance

Why Catching Issues Early Matters

Fixing a vulnerability in production can cost **10–30 times more** than fixing it during development. An SQL injection caught after deployment might mean weeks of remediation, patches, and potential downtime. The same issue flagged in a developer's editor or local environment could be resolved in minutes. Current tools like CodeQL and SonarQube detect vulnerabilities, but only after developers submit pull requests. When security issues are flagged, the development cycle stalls, PRs fail, developer's context-switch to fix code they wrote days earlier, and the process repeats until all gates pass.

This reactive approach creates friction. After the fact, developers learn about security issues where delivery timelines are extended due to security bottlenecks, and teams miss opportunities to build secure coding habits during active development.

GenAI agents solve this by moving security analysis into the coding process itself, providing real-time feedback that transforms security from a downstream gate into an upstream enabler of faster, safer development.

GenAI vs. Traditional Static Analysis: Cost and Capability Comparison

Although established tools like CodeQL and SonarQube dominate the static analysis landscape, GenAI-powered solutions like SecureCodeAgent offer compelling advantages in both cost and capability. Traditional tools can cost a lot: $19–$99 per user per month for CodeQL through GitHub Advanced Security (https://www.byteplus.com/en/topic/516886), and SonarQube pricing starts at $150 for the developer edition and €30 monthly for SonarQube Cloud's Team plan (100k lines of code) (https://revpilots.com/pricing/sonarqube-pricing). In contrast, Azure OpenAI API typically charges $5 per million input tokens and $15 per million output tokens (https://learn.microsoft.com/en-us/answers/questions/2107857/cost-rates-and-historical-cost-of-azure-openai-mod), translating to roughly $0.02–$0.05 per code analysis for average functions—potentially 10–50 times cheaper for small to medium teams.

Beyond cost, GenAI excels in areas where rule-based tools struggle: It explains why code is vulnerable rather than just flagging it, applies security knowledge across languages, adapts to new vulnerability patterns without manual rule updates, and provides natural language explanations that improve developer education. Although CodeQL and SonarQube provide comprehensive rule coverage and enterprise-grade reporting, GenAI tools enhance

the developer experience through conversational feedback, instant explanations, and the ability to understand business context, making security more accessible and educational rather than merely punitive.

By moving security checks closer to where developers write code, they can:

- Reduce delays caused by back-and-forth in PR reviews.
- Avoid introducing vulnerabilities into shared branches.
- Improve their own awareness of secure coding practices.

This approach forms the foundation of shift-left security—addressing problems as early as possible.

How SecureCodeAgent Works

SecureCodeAgent integrates seamlessly into a developer's existing workflow, as illustrated in **Figure 2**. Traditionally, security scans are performed **after** a Pull Request (PR) is raised using tools such as SonarQube or CodeQL. This approach delays feedback and often surfaces vulnerabilities late in the development cycle.

With SecureCodeAgent, this process shifts earlier—security checks are performed **as soon as the code is ready for a PR**. The agent leverages the **Azure OpenAI API** to conduct an AI-driven scan that intelligently identifies potential security and performance issues within the source code.

Each finding is presented in a **structured, developer-friendly table**, highlighting:

- The specific insecure or inefficient line of code
- A secure replacement suggestion
- The rationale behind the recommendation
- The severity level (high, medium, or low)

By providing instant insights before code review, SecureCodeAgent empowers developers to **resolve issues proactively**, reducing dependency on licensed tools like SonarQube and CodeQL. This not only saves time and cost but also ensures that code reaching the PR stage is **already security-vetted**, accelerating the overall deployment pipeline.

In essence, the proposed flow introduces **GenAI** (powered by SecureCodeAgent) as an early-stage safeguard—giving developers immediate visibility into vulnerabilities and promoting a more secure, efficient, and automated development lifecycle.

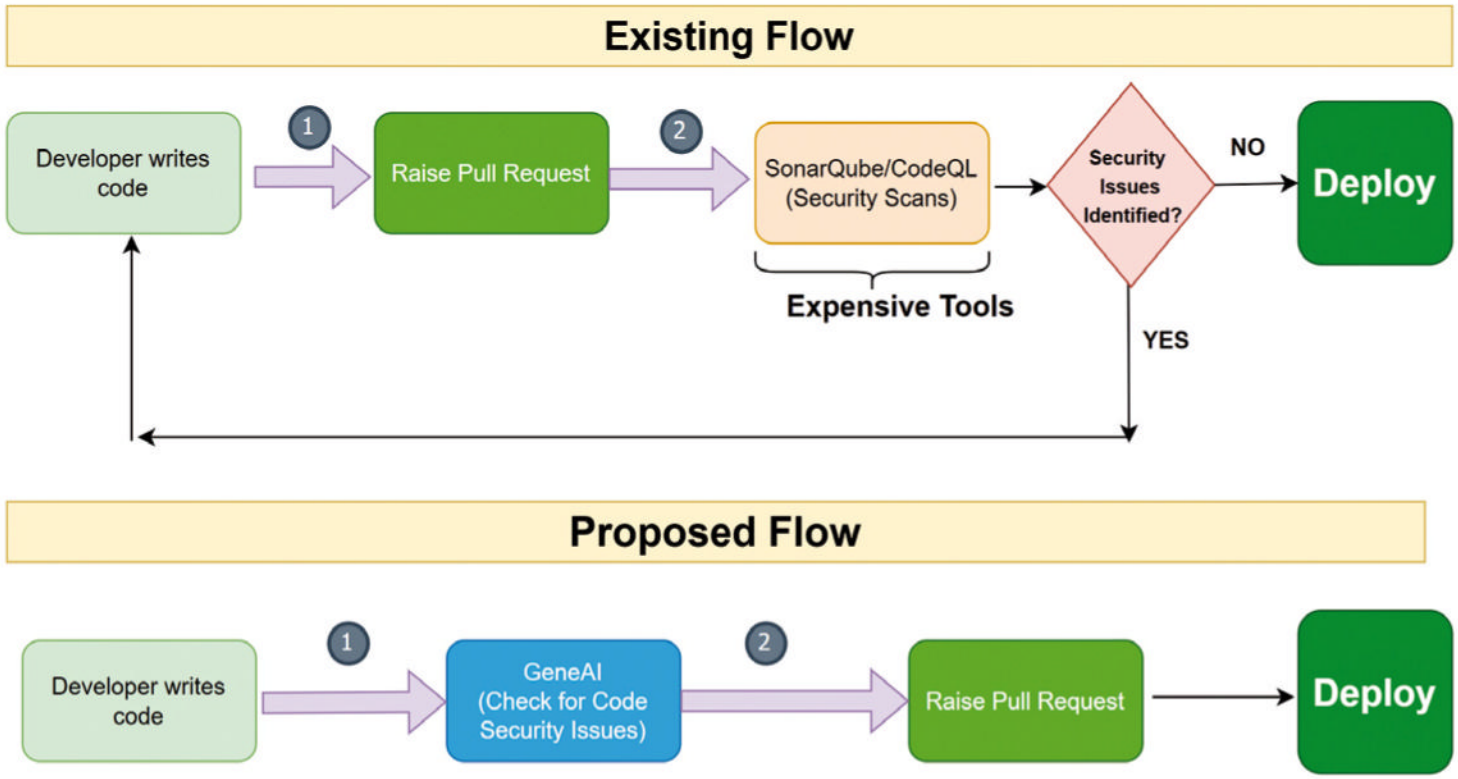


Figure 2: Flow diagram with existing Secure Code Tools vs. SecureCodeAgent

Detected Issue	Suggested Secure Code	Reason for Change	Criticality
Weak hashing (MD5)	Use bcrypt/Argon2 for hashing	MD5 is insecure,vulnerable to brute-force	High

Figure 3: Output generated by running SecureCodeScan on sample code

Listing 1: Invoking SecureCodeScan on Sample Code

```
import os
from openai import AzureOpenAI
# Initialize Azure OpenAI client
client = AzureOpenAI(
  api_key=os.getenv("AZURE_OPENAI_API_KEY"),
  api_version="2024-05-01-preview",
  azure_endpoint=os.getenv("AZURE_OPENAI_ENDPOINT"))
# Example: Developer code snippet to be validated
sample_code = """
import hashlib
def store_password(password):
  # Weak hashing algorithm
  return hashlib.md5(password.encode()).hexdigest()
"""
# Prompt instructs AI to return structured feedback
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
    {
    "role": "system",
      "content": (
        "Analyze the given code and provide results in a table "
        "with columns: 1) Detected Issue, 2) Suggested Secure "
        "Code, 3) Reason for Change, 4) Criticality."
      )
    },
    {
      "role": "user",
      "content": sample_code
    }
  ],
  temperature=0,
  top_p=1
)

# Extract response content
response_content = (
  response.choices[0].message.content
  if response.choices else "No response generated."
)
# Print security analysis
print("=== SecureCodeAgent Analysis ===")
print(response_content)
```

Step-by-Step Demonstration

Listing 1 is an example using Python. The code defines a simple password storage function that uses **MD5 hashing,** a known security weakness. SecureCodeAgent detects the issue and suggests a fix. SecureCodeAgent is demonstrated here as a Python script using the Azure OpenAI API. You can copy this code into your own projects and customize it to their workflow.

Sample Output (AI-generated)

Figure 3 depicts the output generated running **Listing 1** code.

This feedback gives developers instant visibility into security flaws before pushing their code.

Automating the Workflow

Running the script manually is useful, but the real power comes when SecureCodeAgent is integrated into Git workflows.

- **Pre-commit hook:** Run scans before a commit is made.
- **Pre-push hook:** Block insecure code from being pushed upstream.
- **CI/CD pipeline:** Add SecureCodeAgent as a job in GitHub Actions or Jenkins for automated enforcement.

For example, a simple **pre-push Git hook** can automatically run the script and prevent insecure code from reaching the remote repository.

Git Pre-Push Hook with SecureCodeAgent

Save this as .git/hooks/pre-push in your repo and make it executable (chmod +x .git/hooks/pre-push). The following code depicts a bash script for invoking SecureCodeAgent.

```
#!/bin/bash
# Pre-push Git hook to run SecureCodeAgent
# before pushing code
echo "🔍 Running SecureCodeAgent security scan.."
# Path to your Python SecureCodeAgent script
SECURECODE_AGENT_SCRIPT="./securecodeagent_\
scan.py"
# Run the scan
python3 $SECURECODE_AGENT_SCRIPT
# Capture the exit status
STATUS=$?
if [ $STATUS -ne 0 ]; then
  echo "❌ SecureCodeAgent found issues."\
"Push aborted."
  exit 1
else
  echo "✅ No critical security issues found."
   echo " Proceeding with push..."
fi
```

Technical Deep Dive

Before jumping into the implementation, it helps to understand how SecureCodeAgent communicates with the model. The agent relies on structured JSON messages sent to the API, where each message defines the role (system, user) and content (instructions or code). The model then responds in JSON as well, which can be easily parsed back into Python objects.

This approach makes the workflow transparent:

1. **Python code builds the request** (JSON payload with model, messages, and parameters).
2. **The API analyzes the snippet** and returns structured JSON feedback.
3. **Python extracts the response content** from the JSON object.

This design shows how lightweight the integration can be—no external frameworks or SDKs are required beyond the API client itself.

```python
def run_securecodeagent_json(code_snippet):
    """Run SecureCodeAgent and return structured JSON findings."""
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": (
                    "Analyze the given code and respond strictly in "
                    "JSON format with keys: issue, suggestion, reason, "
                    "criticality."
                )
            },
            {"role": "user", "content": code_snippet}
        ],
        temperature=0
    )
    return json.loads(response.choices[0].message.content)

# Example usage
code_to_test = """
def fetch_data(user_input):
    query = "SELECT * FROM orders WHERE id = '" + user_input + "';"
    return query
"""

results = run_securecodeagent_json(code_to_test)
print("=== Structured Findings ===")
print(json.dumps(results, indent=2))
```

Figure 4: Secure Code Agent response in JSON format

With that context in mind, let's look at a pure Python example that demonstrates how JSON-based requests and responses power SecureCodeAgent in practice.

Pure Python Example

SecureCodeAgent can also return structured JSON output. For a code example, refer to **Figure 4**. This code makes it easy to integrate into existing testing frameworks or dashboards.

Error Handling and Robustness

When integrating with Azure OpenAI APIs, robustness is essential. Developers don't want their workflow to break if an API call fails. Common issues include rate limits, timeouts, or malformed responses. By adding retry logic with exponential backoff, SecureCodeAgent can handle these gracefully.

The following code is an example of retry logic for SecureCodeAgent:

```python
def analyze_with_retry(sample_code, max_retries=3):
 for attempt in range(max_retries):
  try:
   resp = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
     {
      "role": "system",
      "content": "Analyze code for security
         Issues."
     },
     {
      "role": "user",
      "content": sample_code
     }
    ],
    temperature=0
    )
   return resp.choices[0].message.content
  except RateLimitError:
   time.sleep(2 ** attempt)
  except Exception as e:
   logging.error(e)
```

This ensures that SecureCodeAgent remains **non-disruptive**: Even if the API is unavailable, developers see a clear message instead of a crash.

Multi-Language Support

Although the example shows Python, the approach works equally for Java, JavaScript, and other languages. Devel-

opers can pass any code snippet to the API, and SecureCodeAgent provides context-aware recommendations. A future enhancement could involve automatically detecting the language and applying tailored checks per ecosystem.

Performance Considerations

Running AI scans on every commit could feel heavy if not optimized. A practical approach is to limit checks to files modified in the current commit or PR. Developers can also configure SecureCodeAgent to only block pushes for high severity issues, and logging medium/low ones as warnings. This keeps security proactive but non-intrusive.

Managing False Positives

No static or AI-powered tool is perfect. SecureCodeAgent should allow developers to mark findings as "acknowledged" or "ignored" for cases where the code is acceptable. Storing these decisions in a configuration file (e.g., .securecodeignore) ensures that teams don't waste time triaging the same false positives repeatedly.

Benefits and Lessons Learned

By shifting security left, SecureCodeAgent helps:

- **Developers**: Catch issues early without leaving their editor
- **AppSec teams**: Focus on higher-level security rather than routine issues
- **Organizations**: Reduce costly post-deployment fixes

Industry evidence demonstrates the value of this approach: Shifting security from the Build stage to the earlier Local Development and Code Commit stages improves security, while saving time, increasing productivity, and enhancing developer experience.-

Conclusion

Developers can no longer treat security as an afterthought in modern software development. By combining the power of GenAI with developer-friendly workflows, SecureCodeAgent enables security by design. Developers receive real-time insights into their code, teams reduce costly rework, and organizations deliver software faster and safer.

The takeaway is simple: Shifting security to the development stage benefits everyone.

Gaurav Mittal

CODE

Nov/Dec 2025
Volume 26 Issue 6

Group Publisher
Markus Egger

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers in This Issue
Enzo Grubisa
Joydip Kanjilal
Vassili Kaplan
Sonu Kapoor
Sahil Malik
Nevio Medancic
Gaurav Mittal
Jason Murphy
Paul Sheriff

Technical Reviewers
Markus Egger
Rod Paddock

Production
Friedl Raffeiner Grafik Studio
www.frigraf.it

Graphic Layout
Friedl Raffeiner Grafik Studio in collaboration with onsight (www.onsightdesign.info)

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext. 26
tammy@code-magazine.com

Circulation & Distribution
General Circulation: EPS Software Corp.
Newsstand: Accelerate360
EBSCO
Magzter
Zinio

Subscriptions

Circulation Manager
Colleen Cade
832-717-4445 ext. 28
ccade@codemag.com

US subscriptions are $29.99 USD for one year.
Subscriptions outside the US are $50.99 USD.
Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa and Discover credit cards accepted.
Back issues are available. For subscription information, email subscriptions@code-magazine.com or contact customer service at 832-717-4445 ext. 9.

Subscribe online at
www.code-magazine.com
CODE Developer Magazine
EPS Software Corporation / Publishing Division
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379 USA
Phone: 832-717-4445

YOUR PARTNER FOR CUSTOM SOFTWARE SOLUTIONS

REAL BUSINESS VALUE FOR AI

TRAINING / MENTORING

CUSTOM APPLICATION DEVELOPMENT

CONTINGENT IT STAFFING

Is your development team struggling to complete business-critical projects on time? Are you looking to harness cutting-edge technologies, including AI, for maximum impact?

CODE EXCELS IN:

- **AI integration for enhanced functionality**
- **.NET web and desktop development**
- **Azure cloud migration and transformation**
- **Blazor development**
- **Mobile app creation**
- **Staffing, training and mentoring**

Let CODE transform your software challenges into competitive advantages.

CONTACT US TODAY. NO STRINGS. NO COMMITMENT.

codemag.com/code

832-717-4445 ext. 9 • info@codemag.com

©shutterstock

CODE
STAFFING

UNLOCK STAFFING EXCELLENCE

Top-Notch IT Talent, Contract Flexibility, Happy Teams, and a Commitment to Customer Success Converge with CODE Staffing

Our IT staffing solutions are engineered to drive your business forward while saving you time and money. Say goodbye to excessive overhead costs and lengthy recruitment efforts. With CODE Staffing, you'll benefit from contract flexibility that caters to both project-based and permanent placements. We optimize your workforce strategy, ensuring a perfect fit for every role and helping you achieve continued operational excellence.

Ready to Discuss Your IT Staffing Needs?

Visit our website to find out more about how we are changing the staffing industry.

Website: codestaffing.com
Yair Alan Griver (yag)
Chief Executive Officer
Direct: +1 425 301 1590
Email: yag@codestaffing.com